DOI 10.1007/s00165-016-0363-5

Formal Aspects of Computing (2016) 28: 643–667

Formal Aspects of Computing

Dynamic role authorization in multiparty conversations

Silvia Ghilezan[1], Svetlana Jakšić[1], Jovanka Pantović[1], Jorge A. Pérez[2], Hugo Torres Vieira[3]

[1] University of Novi Sad, Novi Sad, Serbia
[2] University of Groningen, Groningen, The Netherlands
[3] IMT School for Advanced Studies Lucca, Lucca, Italy

Abstract. Protocols in distributed settings usually rely on the interaction of several parties and often identify the *roles* involved in communications. Roles may have a behavioral interpretation, as they do not necessarily correspond to sites or physical devices. Notions of *role authorization* thus become necessary to consider settings in which, e.g., different sites may be authorized to act on behalf of a single role, or in which one site may be authorized to act on behalf of different roles. This flexibility must be equipped with ways of controlling the roles that the different parties are authorized to represent, including the challenging case in which role authorizations are determined only at runtime. We present a typed framework for the analysis of multiparty interaction with dynamic role authorization and delegation. Building on previous work on conversation types with role assignment, our formal model is based on an extension of the π-calculus in which the basic resources are pairs channel-role, which denote the access right of interacting along a given channel representing the given role. To specify dynamic authorization control, our process model includes (1) a novel scoping construct for authorization domains, and (2) communication primitives for authorizations, which allow to pass around authorizations to act on a given channel. An authorization error then corresponds to an action involving a channel and a role not enclosed by an appropriate authorization scope. We introduce a typing discipline that ensures that processes never reduce to authorization errors, including when parties dynamically acquire authorizations.

Keywords: Software verification, Type systems, Behavioral types, Process calculi, Authorization control

Correspondence and offprint requests to: Jorge A. Pérez, E-mail: j.a.perez@rug.nl; Hugo Torres Vieira, E-mail: hugotvieira@imtlucca.it

1. Introduction

1.1. Context

Distributed systems operation is based on cooperating remote parties that communicate with each other to coordinate their local actions. In order to complete the tasks they were designed to perform, it is crucial that such interacting parties follow well-defined interaction protocols, as otherwise message contents may be confused and/or systems may get stuck. Furthermore, for the sake of accountability, it is often important to know who are the parties or, more generically, which are the *roles* involved in the interactions. This identification of roles also seems crucial when reasoning about security policies, as such policies are typically expressed in terms of the capabilities of the involved parties.

Different notions of role-based specifications can be found in modern distributed information systems, ranging from access control to structured interactions in communication-centered systems. These notions sometimes build on the assumption that distinct participants (e.g., users at different physical locations) may belong to the same role, and that a single participant may implement several different roles. The fact that roles may be distributed among parties seems particularly relevant, for instance, in scenarios where several worker threads may be able to impersonate a single server role in order to make load balancing transparent to a client service. Similarly, the fact that a single peer can carry out actions associated to different roles seems essential to implement, e.g., a service broker that interacts as a client with other providers while it acts as a provider with clients. Each role may be associated with a set of permissions (e.g., privileges to access data or perform some action), thus enforcing an assignment of permissions to involved participants.

In this work, we address the issue of dynamic role authorizations in multiparty interactions from the perspective of formal models of communication equipped with *behavioral types* [HLV+16]. We introduce a simple process model of communicating systems in which roles and role communication are first-class constructs so as to allow to specify authorization handling in a dedicated way. We equip our model with a type system that ensures that processes faithfully implement multiparty communication protocols, but also that actions of well-typed systems conform to declared role authorizations.

In more detail, we consider communication-centered systems where interaction is governed by roles in the above sense, that is, a role may be carried out by several peers and a single peer may carry out actions on behalf of different roles. Building upon a minimal extension of the π-calculus [SW01], we explore this notion in a setting where interaction is defined by actions performed on communication channels (as usual), but in which each action is associated to a specific role. We then consider that using a communication channel under a specific role is the basic communication capability.

In the model that we propose here, it is essential to ensure that communication capabilities are carried out by parties that are actually granted to do so. This is particularly important when considering that access to channels may be dynamically acquired, and that roles may be flexibly implemented in the system. In order to control resource usage, we introduce *authorizations* at the level of first-class language primitives: this enables us to explicitly specify which parts of the system are authorized to manipulate a (communication) resource and to identify as errors those systems in which (active) communication actions are not duly authorized. Given the system specification language, our typing system ensures, on the one hand, that parties follow well-defined protocols of interaction and, on the other hand, that active communication actions are always properly authorized. Next, we motivate our development on top of role-based interacting systems so as to capture and control authorizations for communication capabilities.

1.2. Motivation

We motivate our development by discussing the underlying principles, using illustrative examples. We focus our discussion on features added for the sake of authorization handling and do not further motivate here our base model. In particular, given our intent to model and discipline multiparty interaction, message tags are necessary to ensure linear interaction on a communication medium shared by multiple parties (see [CV10]). Also, for the purpose of (flexibly) handling role-based protocol specifications, we associate roles to communication actions and (for now) do not address scenarios where roles may be dynamically acquired via communications (see [BCVV12]).

Resources in a communication-centered setting. We focus on communication-centered systems where the notion of *resource* is related to communication capabilities. Let a and $\mathbf{r}$ denote a channel and a role, respectively. Given our interest in role-based communication protocols carried out in channels, we identify the pair channel-role, denoted by $(a, \mathbf{r})$, as the basic resource associated to the capability of communicating along a under role $\mathbf{r}$. For instance, we write $(chat, \mathbf{alice})$ to refer to the capability of sending or receiving messages along channel *chat* under role **alice**.

Authorization domain. An authorization to act upon a resource may be described in a natural way by identifying the part(s) of a system in which the authorization holds (or is valid at a given time), hence in which the resource may be used. We thus specify authorizations by specifying their *scope* and the resource to which they refer. Since in our setting resources are channel-role pairs, we specify this notion linguistically via a *scoping operator*: we write $(a, \mathbf{r})P$ to denote that (sub)system P is authorized to act as $\mathbf{r}$ in channel a. For example, we may write $(chat, \mathbf{alice})P \mid (chat, \mathbf{bob})Q$ to capture an (authorized) interaction along channel *chat* between processes P and Q, which may act as roles **alice** and **bob**, respectively. This allows us to distinguish the authorization from the resource itself: knowing (or discovering) the identities of channels and roles does not directly convey the authorization to use them. Notice that in $(a, \mathbf{r})P$ neither a nor $\mathbf{r}$ are bound, differently from other scoping operators, such as channel restriction $(\nu a)P$.

Under this understanding of authorization scope, we may interpret process $(a, \mathbf{r})(P \mid Q)$ as a system in which *both* P and Q are authorized on the pair $(a, \mathbf{r})$ and, given the similarity of the informal readings, we expect such a specification to be equivalent to $(a, \mathbf{r})P \mid (a, \mathbf{r})Q$—the formal meaning of this equivalence shall be made clear later on. We may thus represent, for example, a pool of threads where each one is authorized to act on a given channel role pair—consider, e.g., a service provider that for the sake of responsiveness has several copies of the service, all of them authorized to act on behalf of the service provider.

Dynamic authorization creation. We uniformly integrate authorization scopes in our process specification language. This allows us to model top-level authorizations but also to specify authorization scopes at any other level of systems or subsystems, for instance at the level of the individual branches of a parallel composition or at the level of the continuation of a communication action. Hence, by $\sigma.(a, \mathbf{r})P$ we represent a system that is willing to perform action σ, and then evolve to a system that is authorized on $(a, \mathbf{r})$—which we may call dynamic authorization scope creation. This seems the most sensible choice once we consider dynamic channel name creation (via name restriction), for which there must be a means to specify the respective authorizations—consider, e.g., the system $\sigma.(\nu a)(a, \mathbf{r})P$ where after performing α a new name a is created, and an authorization scope $(a, \mathbf{r})$ is specified.

Channel passing. In an authorization-free setting all channel-role pairs are authorized by default; the ability to dynamically create authorizations may be also useful to enhance this usual setting. This may be important to describe "public" interactions or, more interestingly, role-based authorizations: for instance, a client willing to interact with a service provider may be authorized to do so a priori in some public channel but also in some private channel. As an example, in our model we can specify the system

$$(service, \mathbf{client})service_{\mathbf{client}}?req(x).(x, \mathbf{client})P$$

Intuitively, prefix $service_{\mathbf{client}}?req(x)$ denotes the reception of a channel (x) in message tag req, along channel $service$ with role **client**. In the above process, this prefix is followed by an authorization to interact as **client** in the received channel x. The outermost authorization scope therefore directly concerns the reception along $service$. Notice that regardless of the identity of the channel received in the message the (sub)system in scope of the authorization (P) is authorized to interact on such a channel under role **client**.

Authorization communication. Statically specifying authorizations does not suffice in general, and we are interested in controlling authorizations in a more fine-grained way. Namely, we might not want to specify that a given role is authorized for any possible received channel. In cases in which statically prescribing authorizations is inconvenient, it is crucial to have a mechanism to *communicate* authorizations to other parties. Consider, e.g., when a server delegates a task to a worker thread, which may need to impersonate the server for carrying out its task: the server should be able to pass along the authorization for its role in a *specific* channel. To this end, we introduce primitives for communicating authorizations: given a channel a, a message tag l, and roles $\mathbf{r}$ and

$\mathbf{s}$, prefixes for sending and receiving authorizations will be denoted $a_{\mathbf{s}}l\langle\mathbf{s}\rangle$ and $a_{\mathbf{r}}l(\mathbf{s})$, respectively. As explained next, the communication of a role $\mathbf{r}$ will represent the transfer of the authorization to act as $\mathbf{r}$.

We distinguish authorization communication from usual channel passing much like we distinguish an authorization from the resource itself, hence passing along an identity does not imply passing along the authorization. This allows us to specify scenarios where the name passing infrastructure may include unverified parties through which channel identities may be passed along, but where such parties do not gain authorization to use the channels —see the *broker* in the motivating example below.

In our target setting, systems are specified by considering that roles are statically determined, i.e., there is no dynamic role discovery. As such, communicating the authorization to act on behalf of a role does not involve communicating the identity of the role, and a party willing to receive an authorization on a given role must already know its identity. Hence, when we write, e.g., $service_{\mathbf{worker}}auth(\mathbf{server}).P$ to specify the reception (by a $\mathbf{worker}$) of an authorization to act as $\mathbf{server}$, this role is already determined: this prefix represents an explicit authorization request to act as $\mathbf{server}$. This is different from the name reception prefix, since the object of an authorization reception prefix is not a binder. We write $service_{\mathbf{master}}auth\langle\mathbf{server}\rangle.P$ to specify the (dual) emission (by a $\mathbf{master}$) of an authorization to act as $\mathbf{server}$. In order to send an authorization we view as a necessary condition that the emitting party possesses the communicated authorization.

Authorization delegation. We view authorizations as a resource that should be accountable in some way, rather than as an unrestricted (unlimited) resource. For this purpose, we introduce a notion of delegation associated to authorization communication, in order to represent transference of (accountable) resources. Thus, whenever an authorization is communicated the emitting party loses it. In such a way we introduce a notion of authorization accounting in our model, since, intuitively, we may distinguish between $P_1 = (a, \mathbf{r})Q$ and $P_2 = (a, \mathbf{r})(a, \mathbf{r})Q$: indeed, while P_2 may potentially delegate the authorization twice, P_1 may do it only once. However, since we address concurrent systems that share authorizations, we associate the delegation directly to the "thread" performing the authorization communication. This way, for instance, if a process P is willing to delegate the authorization on $(a, \mathbf{r})$ such a delegation action will not interfere with process Q if both P and Q share the authorization (regardless if we write the system as $(a, \mathbf{r})P \mid (a, \mathbf{r})Q$ or $(a, \mathbf{r})(P \mid Q)$).

1.3. Motivating example

We now illustrate the novel constructs of our model in a concrete example. Consider a system where a client wishes to establish an (authorized) interaction with a service provider, mediated by a broker which is involved in the channel passing but is not authorized to interact through it. We also consider that the service provider outsources a specific task to a third-party provider. We may write

$$Client \mid Broker \mid (vchat)(chat, \mathbf{server})Provider \mid OtherProvider$$

where the channel used in the interaction between client and server ($chat$) is originally held by the service provider, which also has an authorization to interact in the channel as role $\mathbf{server}$. We write

$$Provider \triangleq (service, \mathbf{server})service_{\mathbf{server}}!offer(chat).chat_{\mathbf{server}}!hello().P$$

to say that the provider is willing to share channel $chat$ in message $offer$ (using the authorized $service_{\mathbf{server}}$), after which sending a greeting message ($hello$) in channel $chat$, role $\mathbf{server}$. This will allow name $chat$ to be shared with others that might then receive the greeting message. As for the broker, we write

$$Broker \triangleq (service, \mathbf{broker})(brokerservice, \mathbf{broker})service_{\mathbf{broker}}?offer(x).brokerservice_{\mathbf{broker}}!offer(x)$$

so as to implement a forwarding behavior (essentially it receives a channel from the server and forwards in a different channel to the client). Notice that the broker is not authorized to interact in the received channel x, even if it will get to know its identity. We may then specify the client

$$Client \triangleq (brokerservice, \mathbf{client})brokerservice_{\mathbf{client}}?offer(x).(x, \mathbf{client})x_{\mathbf{client}}?hello().Q$$

which, differently from the broker, includes an authorization to act (as role $\mathbf{client}$) in the received channel. Then, according to the operational semantics of our model, we will have that the whole system above evolves in one

step to

$$\begin{array}{l} Client \\ \mid (vchat)((service, \mathbf{broker})(brokerservice, \mathbf{broker})brokerservice_{\mathbf{broker}}!offer(chat) \\ \qquad\qquad \mid (chat, \mathbf{server})(service, \mathbf{server})chat_{\mathbf{server}}!hello().P) \\ \mid OtherProvider \end{array}$$

where *chat* is communicated from the provider to the broker, and in another step to

$$\begin{array}{l} (vchat)((brokerservice, \mathbf{client})(chat, \mathbf{client})chat_{\mathbf{client}}?hello().Q \\ \qquad\qquad \mid (chat, \mathbf{server})(service, \mathbf{server})chat_{\mathbf{server}}!hello().P) \\ \mid OtherProvider \end{array}$$

where *chat* is communicated from the broker to the client (we omit the broker at this point since its contribution to the interaction has finished). Notice the authorization the client specified for the received name now specifies the *chat* channel, so the interaction in *chat* can take place leading to

$$\begin{array}{l} (vchat)((brokerservice, \mathbf{client})(chat, \mathbf{client})Q \\ \qquad\qquad \mid (chat, \mathbf{server})(service, \mathbf{server})P) \\ \mid OtherProvider \end{array}$$

At this point let us consider that the provider wishes to delegate a task to the third party provider which involves impersonating **server** in the *chat* conversation. We may then specify

$$P \triangleq service_{\mathbf{server}}!delegate(chat).(chat, \mathbf{master})chat_{\mathbf{master}}auth\langle\mathbf{server}\rangle.P'$$

and

$$OtherProvider \triangleq (service, \mathbf{slave})service_{\mathbf{slave}}?delegate(y).(y, \mathbf{slave})y_{\mathbf{slave}}auth(\mathbf{server}).y_{\mathbf{server}}!finalize()$$

where, apart from the channel passing there is also a step that involves authorization communication. In order to impersonate **server** in channel *chat*, the third party provider first receives the channel (instantiating y) and after which receives an authorization in the respective channel (message *auth*), which then allows for an authorized interaction in the received channel, acting as **server**, sending message $finalize$. Notice that while the third party provider receives the authorization, the continuation process P' on the original service provider side loses the communicated authorization.

1.4. Structure of the paper

The rest of this paper is organized as follows. In Sect. 2, we define our process language and further illustrate the intended model of dynamic role authorization. The language is endowed with two operational semantics, a reduction semantics and a Labeled Transition System (LTS), which are shown to coincide. Section 3 presents the type system and the properties for well-typed processes. In Sect. 4, we comment on related works. Finally, Sect. 5 draws some concluding remarks and discusses open problems. Omitted proofs have been collected in the Appendix.

This paper offers a unified presentation of two papers: [GJP+14] and [GJP+15]. While the paper [GJP+14] already considered dynamic role authorization in the setting of conversation types, the paper [GJP+15] introduced the idea of authorization scopes to control and give a spatial meaning to explicit authorizations on process actions which inspires the work presented in this document. As such, the papers [GJP+14] and [GJP+15] offer orthogonal perspectives to the issue of dynamic role authorization with delegation; the current paper combines these two perspectives in a uniform way. In this presentation, we have added a Labeled Transition System supporting authorization (not present in [GJP+14, GJP+15]), together with a preliminary investigation on its associated behavioral theory, and have extended the type system (based on conversation types) in order to account for explicit role authorization and authorization scopes.

2. Process language

In this section we present our process language for dynamic role authorization. Our language arises as an extension of the π-calculus, and is endowed with a reduction semantics and a labeled transition system (LTS).

Table 1. Syntax of processes

P	::=	0	(Inaction)	σ	::=	$a_{\mathbf{s}}!l(b)$	(Output)
	|	$P \mid P$	(Parallel)	|	$a_{\mathbf{r}}?l(x)$	(Input)
	|	$(\nu a)P$	(Restriction)	|	$a_{\mathbf{s}}\, l\langle \mathbf{d}\rangle$	(Send authorization)
	|	$(a, \mathbf{r})P$	(Authorization scope)	|	$a_{\mathbf{r}}\, l(\mathbf{d})$	(Receive authorization)
	|	$\sigma.P$	(Prefix)				

Table 2. Structural congruence

(SC-ALPHA)	$P \equiv_\alpha Q \Rightarrow P \equiv Q$	(SC-AUTH-INACT)	$(a, \mathbf{r})0 \equiv 0$
(SC-PAR-ID)	$P \mid 0 \equiv P$	(SC-AUTH-SWAP)	$(a, \mathbf{r})(b, \mathbf{s})P \equiv (b, \mathbf{s})(a, \mathbf{r})P$
(SC-PAR-COMM)	$P \mid Q \equiv Q \mid P$	(SC-AUTH-DIST)	$(a, \mathbf{r})(P \mid Q) \equiv (a, \mathbf{r})P \mid (a, \mathbf{r})Q$
(SC-PAR-ASSOC)	$(P \mid Q) \mid R \equiv P \mid (Q \mid R)$	(SC-AUTH-SCOPE)	$(a, \mathbf{r})(\nu b)P \equiv (\nu b)(a, \mathbf{r})P \quad \text{if } a \neq b$
(SC-REST-INACT)	$(\nu a)0 \equiv 0$		
(SC-REST-EXTR)	$P \mid (\nu a)Q \equiv (\nu a)(P \mid Q) \quad \text{if } a \notin \mathsf{fn}(P)$		
(SC-REST-COMM)	$(\nu a)(\nu b)P \equiv (\nu b)(\nu a)P$		

2.1. Syntax

We consider a synchronous π-calculus [SW01] extended with tagged communications and prefixes for authorization sending and receiving. Let $\mathcal{L}$, $\mathcal{R}$, and $\mathcal{N}$ be infinite base sets of *tags*, *roles*, and *channels*, respectively. We use $l, l', \ldots$ to range over $\mathcal{L}$; $\mathbf{r}, \mathbf{s}, \mathbf{d}, \mathbf{t} \ldots$ to range over $\mathcal{R}$; and $a, b, c, \ldots$ to range over $\mathcal{N}$.

The syntax of processes is given in Table 1. It includes standard constructs for inaction (0), parallel composition ($P \mid Q$), and name restriction ($(\nu a)P$). We write $(\nu \vec{a})$ as a shorthand for $(\nu a_1) \cdots (\nu a_n)$. To specify communication of channels and authorizations, our language includes four kinds of prefixes, denoted σ. Each prefix is associated to a role $\mathbf{r}$: intuitively, this says that σ is allowed to perform its associated action under $\mathbf{r}$. The intuitive semantics for prefixes follows:

- $a_{\mathbf{s}}!l(b)$ expresses sending of name b, in a message with tag l, along channel a, under role $\mathbf{s}$;
- $a_{\mathbf{r}}?l(b)$ expresses receiving of name b, in a message with tag l, along channel a, under role $\mathbf{r}$.

These two prefixes are taken from [BCVV12]. The second pair of prefixes is new to our calculus and concerns the communication of an authorization to act on a role:

- $a_{\mathbf{s}}\, l\langle \mathbf{d}\rangle$ expresses sending authorization for role $\mathbf{d}$, in a message with tag l, along channel a, *under role* $\mathbf{s}$;
- $a_{\mathbf{r}}\, l(\mathbf{d})$ expresses receiving authorization for role $\mathbf{d}$, in a message with tag l, along channel a, *under role* $\mathbf{r}$.

In addition to these new prefixes, we introduce a new scoping construct:

- $(a, \mathbf{r})P$ specifies the scope of authorizations: it authorizes all actions on channel a in P to act according to role $\mathbf{r}$ (up to delegation).

The set of free names of these novel constructs for authorization manipulation is defined as follows:

$$\mathsf{fn}((a, \mathbf{r})P) = \mathsf{fn}(a_{\mathbf{r}}\, l(\mathbf{d})P) = \mathsf{fn}(a_{\mathbf{r}}\, l\langle \mathbf{d}\rangle P) \triangleq \mathsf{fn}(P) \cup \{a\}.$$

2.2. Reduction semantics

The process semantics is defined via a reduction relation, which is defined in Table 3 and denoted $\rightarrow$. Reduction is closed under static contexts and *structural congruence*, denoted $\equiv$, defined in Table 2 following standard lines (cf. [SW01]). Apart from the standard identities for the static fragment of the π-calculus (cf. Table 2, left column), structural congruence gives basic principles for the novel authorization scope (cf. Table 2, right column):

Table 3. Reduction relation. δ abbreviates a sequence of authorization scopes $(a_1, \mathbf{r}_1)\cdots(a_k, \mathbf{r}_k)$

(COMM)

$$\delta_1\,(b, \mathbf{s})b_{\mathbf{s}}!l(c).P \mid \delta_2\,(b, \mathbf{r})b_{\mathbf{r}}?l(x).Q \to \delta_1\,(b, \mathbf{s})P \mid \delta_2\,(b, \mathbf{r})Q\{c/x\}$$

(AUTH)

$$\delta_1\,(b, \mathbf{s})(b, \mathbf{d})b_{\mathbf{s}}l\langle\mathbf{d}\rangle.P \mid \delta_2\,(b, \mathbf{r})b_{\mathbf{r}}l(\mathbf{d}).Q \to \delta_1\,(b, \mathbf{s})P \mid \delta_2\,(b, \mathbf{r})(b, \mathbf{d})Q$$

(PARC) $\dfrac{P \to Q}{P \mid R \to Q \mid R}$ (NEWC) $\dfrac{P \to Q}{(\nu a)P \to (\nu a)Q}$ (STRU) $\dfrac{P \equiv P' \to Q' \equiv Q}{P \to Q}$

(1) authorizations can be discarded/created only for the inactive process (SC-AUTH-INACT);

(2) authorizations can be swapped around (SC-AUTH-SWAP);

(3) authorizations distributes over parallel composition (SC-AUTH-DIST); and

(4) authorizations and name restrictions can be swapped if the corresponding names differ (SC-AUTH-SCOPE).

We remark that, differently from name restrictions, authorization scopes can neither be extruded/confined: since authorizations are specified for free names (that cannot be α-converted), extruding/confining authorizations actually changes the meaning of processes. For example, we have

$$a_{\mathbf{s}}!l_1(b).0 \mid (a, \mathbf{s})\,0 \not\equiv (a, \mathbf{s})(a_{\mathbf{s}}!l_1(b).0 \mid 0)$$

because both parallel branches of the right-hand side process are authorized to interact on a (role $\mathbf{s}$), while one of the parallel branches of the left-hand side process is not (we return to this issue in Example 3). Another distinctive property comes from the significance of multiplicity of authorization scopes. We do not adopt

$$(a, \mathbf{r})P \equiv (a, \mathbf{r})(a, \mathbf{r})P \text{ for } P \neq 0, \tag{1}$$

for the sake of authorization accounting. Before presenting the operational semantics of the language, we ensure that the rewriting supported by structural congruence is enough to isolate top-level communication actions together with their respective authorization scopes. We require some extra notation: we write δ as an abbreviation of a sequence of authorization scopes $(a_1, \mathbf{r}_1)\cdots(a_k, \mathbf{r}_k)$. This way, a process $(a_1, \mathbf{r}_1)\cdots(a_k, \mathbf{r}_k)P$ can be simply written as $\delta\,P$. With a slight abuse of notation, we write $(a_j, \mathbf{r}_j) \in \delta$ if the scope $(a_j, \mathbf{r}_j)$ is included in δ.

Proposition 1 (Normal form) *For any process Q, there are $P_1, \ldots, P_k$, $\sigma_1, \ldots, \sigma_k$, $\vec{c}$, and $\delta_1, \ldots, \delta_k$, where $(\nu\vec{c})$ and δ_i for $i \in 1, \ldots, k$ can be empty sequences, such that*

$$Q \equiv (\nu\vec{c})(\delta_1\,\sigma_1.P_1 \mid \delta_2\,\sigma_2.P_2 \mid \ldots \mid \delta_k\,\sigma_k.P_k) \tag{2}$$

Proof (by induction on the structure of Q). □

We now describe the reduction rules in Table 3. Rules (COMM) and (AUTH) specify synchronizations: two processes can exchange a message l on a channel b under roles $\mathbf{s}$ and $\mathbf{r}$, only if they are authorized for the specified roles on the channel b. Using Rule (AUTH), a process can authorize another process to act under a role $(\mathbf{d})$ only if the first process has permission to do such an authorization (it is in the scope of $(b, \mathbf{d})$) and the second process is asking for explicit authorization of the same role $(\mathbf{d})$. As the considered role will be unauthorized in the continuation of the sender process, we consider this interaction as a kind of *authorization delegation*. Rules (PARC), (NEWC), (STRU) are standard and/or self-explanatory.

We use $\to^*$ to denote the reflexive and transitive closure of $\to$. We introduce some auxiliary notions in order to syntactically characterize *authorization errors* in our setting. First of all, we define the usual notion of *active contexts* for our calculus:

Definition 1 (*Active context*) $\mathcal{C}[\cdot] ::= \cdot \mid P \mid \mathcal{C}[\cdot] \mid (\nu a)\mathcal{C}[\cdot] \mid (a, \mathbf{r})\mathcal{C}[\cdot]$

Active contexts allow us to talk about any active communication prefixes of a process. Also, we may introduce a predicate that states that an active context authorizes actions on a given channel. More precisely, for a given context and a channel-role pair, when the hole of the context is filled with an action on the channel, it is authorized for that action.

Definition 2 (*Context authorization*) For an active context $\mathcal{C}[\cdot]$ and a pair $(a, \mathbf{r})$, the context authorization predicate, denoted $auth(\mathcal{C}[\cdot], (a, \mathbf{r}))$, is defined inductively on the structure of $\mathcal{C}[\cdot]$ as

$$auth(\mathcal{C}[\cdot], (a, \mathbf{r})) \triangleq \begin{cases} false & \text{if } \mathcal{C}[\cdot] = \cdot \\ true & \text{if } \mathcal{C}[\cdot] = (a, \mathbf{r})\mathcal{C}'[\cdot] \\ auth(\mathcal{C}'[\cdot], (a, \mathbf{r})) & \text{if } \mathcal{C}[\cdot] = (b, \mathbf{s})\mathcal{C}'[\cdot] \quad \text{and} \quad a \neq b \text{ or } \mathbf{r} \neq \mathbf{s} \\ auth(\mathcal{C}'[\cdot], (a, \mathbf{r})) & \text{if } \mathcal{C}[\cdot] = P \mid \mathcal{C}'[\cdot] \\ auth(\mathcal{C}'[\cdot], (a, \mathbf{r})) & \text{if } \mathcal{C}[\cdot] = (\nu b)\mathcal{C}'[\cdot] \end{cases}$$

We may then use active contexts and context authorization to precisely characterize errors in our model, since active contexts allow us to talk about any communication prefix in the process and the context authorization predicate allows to account for the authorizations granted by the context: processes that have active communication prefixes which are not authorized are *errors*:

Definition 3 (*Error*) We say process P is an *error* if $P \equiv \mathcal{C}[\sigma_{(a,\mathbf{r})}.Q]$ where

1. $auth(\mathcal{C}[\cdot], (a, \mathbf{r})) = false$ or
2. $\sigma_{(a,\mathbf{r})} = a_{\mathbf{r}} l\langle \mathbf{d} \rangle$ and $auth(\mathcal{C}[\cdot], (a, \mathbf{d})) = false$.

We write $\sigma_{(a,\mathbf{r})}$ to denote a communication prefix on channel a under role $\mathbf{r}$, so essentially any communication action that may cause a stuck configuration according to our semantics (where synchronizations only occur when processes hold the proper authorizations) is seen as an error.

We illustrate the reduction relation and our notion of error by means of a sequence of examples:

Example **1** We show two processes that are authorized for all their actions and that safely reduce to 0.

$$\begin{aligned} P &= (a, \mathbf{r})(b, \mathbf{s})a_{\mathbf{r}}?l(x).x_{\mathbf{s}}!l_1(a).0 \mid (a, \mathbf{s})(b, \mathbf{r})a_{\mathbf{s}}!l(b).b_{\mathbf{r}}?l_1(x).0 \\ &\to (a, \mathbf{r})(b, \mathbf{s})b_{\mathbf{s}}!l_1(a).0 \mid (a, \mathbf{s})(b, \mathbf{r})b_{\mathbf{r}}?l_1(x).0 \to (a, \mathbf{r})(b, \mathbf{s})0 \mid (a, \mathbf{s})(b, \mathbf{r})0 \equiv 0 \\ Q &= (b, \mathbf{s})(a, \mathbf{d})b_{\mathbf{s}}!l(a).a_{\mathbf{s}}l_1\langle \mathbf{d} \rangle.0 \mid (b, \mathbf{r})b_{\mathbf{r}}?l(x).(x, \mathbf{r})x_{\mathbf{r}}l_1(d).0 \\ &\to (b, \mathbf{s})(a, \mathbf{d})a_{\mathbf{s}}l_1\langle \mathbf{d} \rangle.0 \mid (b, \mathbf{r})(a, \mathbf{r})a_{\mathbf{r}}l_1(d).0 \to (b, \mathbf{s})0 \mid (b, \mathbf{r})(a, \mathbf{r})(a, \mathbf{d})0 \equiv 0 \end{aligned}$$

Example **2** We can go back now to the property (1) and consider processes

$$\begin{aligned} P &= (a, \mathbf{s})(a, \mathbf{r})(a, \mathbf{r})a_{\mathbf{s}}l\langle \mathbf{r} \rangle.a_{\mathbf{r}}!l_1(b).0 \mid (a, \mathbf{t})a_{\mathbf{t}}l(\mathbf{r}).a_{\mathbf{r}}?l_1(x).0 \text{ and} \\ Q &= (a, \mathbf{s})(a, \mathbf{r})a_{\mathbf{s}}l\langle \mathbf{r} \rangle.a_{\mathbf{r}}!l_1(b).0 \mid (a, \mathbf{t})a_{\mathbf{t}}l(\mathbf{r}).a_{\mathbf{r}}?l_1(x).0. \end{aligned}$$

Process P is well behaved, since

$$P \to (a, \mathbf{s})(a, \mathbf{r})a_{\mathbf{r}}!l_1(b).0 \mid (a, \mathbf{t})(a, \mathbf{r})a_{\mathbf{r}}?l_1(x).0 \to (a, \mathbf{s})(a, \mathbf{r})0 \mid (a, \mathbf{t})(a, \mathbf{r})0 \equiv 0,$$

while Q gets into an error

$$Q \to (a, \mathbf{s})a_{\mathbf{r}}!l_1(b).0 \mid (a, \mathbf{t})(a, \mathbf{r})a_{\mathbf{r}}?l_1(x).0 \equiv \mathcal{C}[a_{\mathbf{r}}!l_1(b).0],$$

where $\mathcal{C}[\cdot] = (a, \mathbf{s})\cdot \mid (a, \mathbf{t})(a, \mathbf{r})a_{\mathbf{r}}?l_1(x).0$ and $auth(\mathcal{C}[\cdot], (a, \mathbf{r})) = false$.

Example **3** Regarding authorization extrusion/confinement consider the following processes

$$\begin{aligned} P &= (b, \mathbf{r})b_{\mathbf{r}}?l(x).x_{\mathbf{s}}!l_1(b).0 \mid (a, \mathbf{s})(b, \mathbf{s})b_{\mathbf{s}}!l(a).0 \\ Q &= (a, \mathbf{s})((b, \mathbf{r})b_{\mathbf{r}}?l(x).x_{\mathbf{s}}!l_1(b).0 \mid (b, \mathbf{s})b_{\mathbf{s}}!l(a).0) \end{aligned}$$

where P differs from Q by an extrusion of authorization $(a, \mathbf{s})$ (or, conversely, confinement). We have that P reduces to an error while Q does not:

$$\begin{aligned} P &\to (b, \mathbf{r})a_{\mathbf{s}}!l_1(b).0 \mid (a, \mathbf{s})(b, \mathbf{s})0 \\ Q &\to (a, \mathbf{s})((b, \mathbf{r})a_{\mathbf{s}}!l_1(b).0 \mid (b, \mathbf{s})0) \end{aligned}$$

This example shows that authorization extrusion is not sound in our model even when considering an expected side condition on the extruded name (notice a does not occur free in the left-hand side of P).

2.3. Labeled transition system (LTS)

We now introduce a labeled transition system (LTS) for our language. Our LTS extends the (early) LTS for the π-calculus by considering a syntax for labels with explicit annotations and rules for representing authorization scopes. Before formally giving the LTS, we describe some intuitions. Consider the following set of labels:

$$\lambda ::= a_{\mathbf{s}}!l(b) \mid a_{\mathbf{r}}?l(b) \mid a_{\mathbf{s}}l\langle \mathbf{d}\rangle \mid a_{\mathbf{r}}l(\mathbf{d}) \mid (\nu a)b_{\mathbf{s}}!l(a) \mid \tau$$

An LTS based on labels λ would be able to describe process evolution in terms of name output and input, delegation and reception of authorization, name extrusions, and the internal action τ. However, it would not allow us to describe the authorization scope under which any of these actions may take place, nor the fact that senders may lose authorization through communication. To this end, we define two modifications:

(a) We annotate labels λ with authorization scopes of the form $(a, \mathbf{t})$, describing that any action along channel a is authorized on role $\mathbf{t}$. As such, we will consider labels of the form

$$(a, \mathbf{t})^i \lambda$$

where λ is as above and the superscript $i \in \{0, 1\}$ in $(a, \mathbf{t})^i$ denotes the number of occurrences of $(a, \mathbf{t})$. As a convention, we will identify $(a, \mathbf{t})^0 \lambda$ with λ, for any not yet authorized action λ.

(b) We distinguish two kinds of internal actions, denoted τ_ω and $\tau_\omega^{(a,\mathbf{d})}$, where ω stands for zero, one, or two pairs of names and roles. Intuitively, a pair $(a, \mathbf{r})$ included in the ω of τ_ω says that such an internal action depends on an action along name a which still requires authorization for role $\mathbf{r}$. This way, for example, ω will become invisible if the two actions leading to the synchronization are both properly authorized. The pair $(a, \mathbf{d})$ in label $\tau_\omega^{(a,\mathbf{d})}$ states that this internal action was performed by communicating (delegating) the authorization for not yet authorized role $\mathbf{d}$ along a. Once again, it is worth noticing that when an invisible action results from the synchronization of dual actions on authorized roles, we will have label τ. When considering closed processes, only the internal action relation $\xrightarrow{\tau}$ agrees with reduction, up to structural congruence (see Sect. 2.4).

Based on the above intuitions, we may now introduce the set of labels for our LTS, ranged over $\alpha, \alpha', \ldots$. We have:

$$\alpha ::= (a, \mathbf{s})^i a_{\mathbf{s}}!l(b) \mid (a, \mathbf{r})^i a_{\mathbf{r}}?l(b) \mid (a, \mathbf{s})^i (a, \mathbf{d})^j a_{\mathbf{s}}l\langle \mathbf{d}\rangle \mid (a, \mathbf{r})^i a_{\mathbf{r}}l(\mathbf{d}) \mid (\nu a)(b, \mathbf{s})^i b_{\mathbf{s}}!l(a) \mid \tau_\omega \mid \tau_\omega^{(a,\mathbf{d})}$$

where $\omega ::= (a, \mathbf{s})^i (a, \mathbf{r})^j$, with $i, j \in \{0, 1\}$. Given a label α, we write $\mathsf{n}(\alpha)$ and $\mathsf{bn}(\alpha)$ to denote its associated sets of names and bound names, respectively. These sets are defined as expected.

Let $\mathcal{P}$ denote the set of all processes. We define the relation $\xrightarrow{\alpha} \subseteq \mathcal{P} \times \mathcal{P}$, for every action α. As a key novelty, we require a set of rules to govern authorization scopes, including authorization delegation which may or may not involve losing the communicated authorization. The transition relation is inductively defined by the following five groups of rules, denoted (I)–(V):

(I) Rules (LOUT), (LOUT-A), (LIN), (LIN-A) describe actions performed under not yet authorized roles.

$$\frac{}{a_{\mathbf{s}}!l(b).P \xrightarrow{a_{\mathbf{s}}!l(b)} P}\ (\text{LOUT}) \qquad \frac{}{a_{\mathbf{s}}l\langle \mathbf{d}\rangle.P \xrightarrow{a_{\mathbf{s}}l\langle \mathbf{d}\rangle} P}\ (\text{LOUT- A}) \qquad \frac{}{a_{\mathbf{r}}?l(x).P \xrightarrow{a_{\mathbf{r}}?l(b)} P\{b/x\}}\ (\text{LIN}) \qquad \frac{}{a_{\mathbf{r}}l(\mathbf{d}).P \xrightarrow{a_{\mathbf{r}}l(\mathbf{d})} (a, \mathbf{d})P}\ (\text{LIN- A})$$

1. Rule (LOUT) expresses that process $a_{\mathbf{s}}!l(b).P$ can send b (in a message with tag l), along a under role $\mathbf{s}$, and evolve to P.
2. Using Rule (LOUT-A), process $a_{\mathbf{s}}l\langle \mathbf{d}\rangle.P$ can send authorization $\mathbf{d}$ over a under the role $\mathbf{s}$, and evolve to P. This ability is enabled for roles $\mathbf{d}$ and $\mathbf{s}$.
3. By Rule (LIN), process $a_{\mathbf{r}}?l(x).P$ is able to receive b along a, under role $\mathbf{r}$, and evolve to $P\{b/x\}$.
4. By Rule (LIN-A), process $a_{\mathbf{r}}l(\mathbf{d}).P$ can get authorization for $\mathbf{d}$ along a and evolve to $(a, \mathbf{d})P$.

(II) Similarly as in the π-calculus, we have Rules (LRES1) and (LRES2), here not depending on included authorization scopes.

$$\text{(LRES1)}\ \frac{P \xrightarrow{\alpha} Q \quad a \notin \mathsf{n}(\alpha)}{(\nu a)P \xrightarrow{\alpha} (\nu a)Q} \qquad \text{(LRES2)}\ \frac{P \xrightarrow{(b,\mathbf{s})^i b_{\mathbf{s}}!l(a)} Q \quad i \in \{0,1\} \quad a \neq b}{(\nu a)P \xrightarrow{(\nu a)(b,\mathbf{s})^i b_{\mathbf{s}}!l(a)} Q}$$

(III) The observable behavior of a process within an authorization scope is governed by Rules (LSCOPE1) – (LSCOPE5).

$$\text{(LSCOPE1)}\ \frac{P \xrightarrow{\alpha} Q \quad \alpha \notin \{a_{\mathbf{s}} l\langle \mathbf{t}\rangle, (a,\mathbf{s})a_{\mathbf{s}} l\langle \mathbf{t}\rangle, \tau_{\omega}^{(a,\mathbf{t})}\}}{(a,\mathbf{t})P \xrightarrow{(a,\mathbf{t})\cdot\alpha} (a,\mathbf{t})Q} \qquad \text{(LSCOPE4)}\ \frac{P \xrightarrow{(a,\mathbf{d})a_{\mathbf{d}} l\langle \mathbf{d}\rangle} Q}{(a,\mathbf{d})P \xrightarrow{(a,\mathbf{d})(a,\mathbf{d})a_{\mathbf{d}} l\langle \mathbf{d}\rangle} (a,\mathbf{d})Q}$$

$$\text{(LSCOPE2)}\ \frac{P \xrightarrow{a_{\mathbf{d}} l\langle \mathbf{d}\rangle} Q}{(a,\mathbf{d})P \xrightarrow{(a,\mathbf{d})a_{\mathbf{d}} l\langle \mathbf{d}\rangle} Q} \qquad \text{(LSCOPE3)}\ \frac{P \xrightarrow{(a,\mathbf{s})^i a_{\mathbf{s}} l\langle \mathbf{d}\rangle} Q \quad \mathbf{s} \neq \mathbf{d} \quad i \in \{0,1\}}{(a,\mathbf{d})P \xrightarrow{(a,\mathbf{d})(a,\mathbf{s})^i a_{\mathbf{s}} l\langle \mathbf{d}\rangle} Q} \qquad \text{(LSCOPE5)}\ \frac{P \xrightarrow{\tau_{\omega}^{(a,\mathbf{d})}} Q}{(a,\mathbf{d})P \xrightarrow{\tau_{\omega}} Q}$$

In Rule (LSCOPE1) we use an auxiliary function that gives authorization to the pair $(a,\mathbf{t})$ in the label α. It is formalized by the *authorization function*, $(a,\mathbf{t})\cdot\alpha$, defined as follows:

α	$\lambda_{(a,\mathbf{t})}$	$\tau_{\omega(a,\mathbf{t})}^{\omega'}$	$(\nu b)a_{\mathbf{t}}!l(b)$ (where $b \neq a$)	otherwise
$(a,\mathbf{t})\cdot\alpha$	$(a,\mathbf{t})\lambda_{(a,\mathbf{t})}$	$\tau_{\omega}^{\omega'}$	$(\nu b)(a,\mathbf{t})a_{\mathbf{t}}!l(b)$	α

where action λ is defined above and $\lambda_{(a,\mathbf{t})}$ denotes that the action is performed under the role $\mathbf{t}$ along a (i.e. $a_{\mathbf{t}}!l(b)$, $a_{\mathbf{t}}?l(b)$, $a_{\mathbf{t}}l\langle\mathbf{d}\rangle$ or $a_{\mathbf{t}}l(\mathbf{d})$). These rules describe the behavior of process $(a,\mathbf{t})P$ provided that P can evolve to Q by performing action α. In three of these rules, $(a,\mathbf{t})P$ loses authorization scope $(a,\mathbf{t})$ after forwarding it, and then evolves to Q.

(a) By Rule (LSCOPE2), a process can forward authorization $\mathbf{d}$ under the same (once authorized) role $\mathbf{d}$. For example:

$$(a,\mathbf{d})a_{\mathbf{d}} l\langle\mathbf{d}\rangle.a_{\mathbf{d}}!l(b)Q \xrightarrow{(a,\mathbf{d})a_{\mathbf{d}} l\langle\mathbf{d}\rangle} a_{\mathbf{d}}!l(b).Q.$$

In this way, process loses its own authorization scope.

(b) By Rule (LSCOPE3), a process can forward authorization $\mathbf{d}$ under different role $\mathbf{s}$. For example:

$$(a,\mathbf{d})a_{\mathbf{s}} l\langle\mathbf{d}\rangle.Q \xrightarrow{(a,\mathbf{d})a_{\mathbf{s}} l\langle\mathbf{d}\rangle} Q,$$

$$(a,\mathbf{d})(a,\mathbf{s})a_{\mathbf{s}} l\langle\mathbf{d}\rangle.Q \xrightarrow{(a,\mathbf{d})a_{\mathbf{s}} l\langle\mathbf{d}\rangle} (a,\mathbf{s})Q.$$

(c) By Rule (LSCOPE5), a process can internally communicate authorized role $\mathbf{d}$ (see examples (3)-(5)). For example:

$$(a,\mathbf{d})(a_{\mathbf{s}} l\langle\mathbf{d}\rangle.P' \mid a_{\mathbf{r}} l(\mathbf{d}).Q') \xrightarrow{\tau_{(a,\mathbf{r}),(a,\mathbf{s})}} P' \mid (a,\mathbf{d})(a,\mathbf{d})Q',$$

$$(a,\mathbf{d})((a,\mathbf{s})a_{\mathbf{s}} l\langle\mathbf{d}\rangle.P' \mid a_{\mathbf{r}} l(\mathbf{d}).Q') \xrightarrow{\tau_{(a,\mathbf{r})}} (a,\mathbf{s})P' \mid (a,\mathbf{d})(a,\mathbf{d})Q',$$

$$(a,\mathbf{d})((a,\mathbf{s})a_{\mathbf{s}} l\langle\mathbf{d}\rangle.P' \mid (a,\mathbf{r})a_{\mathbf{r}} l(\mathbf{d}).Q') \xrightarrow{\tau} (a,\mathbf{s})P' \mid (a,\mathbf{d})(a,\mathbf{d})(a,\mathbf{r})Q'.$$

Notice that only left-hand side threads lose authorization scope $(a,\mathbf{d})$, since they forward it to right-hand side threads. As right-hand side threads were authorized to communicate role $\mathbf{d}$ over a, authorization scopes $(a,\mathbf{d})$ are doubled after the transitions.

In the other two rules, $(a, \mathbf{t})P$ keeps the authorization scope and evolves to $(a, \mathbf{t})Q$:

(d) By Rule (LSCOPE4), a process having double authorization, after passing an authorization, still keeps its copy. For example:

$$(a, \mathbf{d})(a, \mathbf{d})a_{\mathbf{d}}l\langle\mathbf{d}\rangle.a_{\mathbf{d}}!l(b).Q \xrightarrow{(a,\mathbf{d})(a,\mathbf{d})a_{\mathbf{d}}l\langle\mathbf{d}\rangle} (a, \mathbf{d}).a_{\mathbf{d}}!l(b).Q.$$

(e) Rule (LSCOPE1) covers all the cases where α does not include role delegation. For example,

$$(b, \mathbf{s})b_{\mathbf{s}}!l(a).P' \xrightarrow{(b,\mathbf{s})b_{\mathbf{s}}!l(a)} (b, \mathbf{s})P',$$

$$(a, \mathbf{d})b_{\mathbf{s}}!l(a).P' \xrightarrow{b_{\mathbf{s}}!l(a)} (a, \mathbf{d})P',$$

$$(a, \mathbf{r})(a, \mathbf{s})(a, \mathbf{d})(a_{\mathbf{s}}l\langle\mathbf{d}\rangle.P' \mid a_{\mathbf{r}}l(\mathbf{d}).Q') \xrightarrow{\tau} (a, \mathbf{r})(a, \mathbf{s})(P' \mid (a, \mathbf{d})(a, \mathbf{d})Q').$$

(IV) The four Rules (LCOMM1), (LCOMM2), (LAUTH1), and (LAUTH2) capture to internal actions (we omit the four symmetric rules with commuted parallel processes in conclusions of the respective rules).

(LCOMM1)
$$\frac{P \xrightarrow{\alpha_{\omega_1}} P' \quad Q \xrightarrow{\overline{\alpha}_{\omega_2}} Q'}{P \mid Q \xrightarrow{\tau_{\omega_1\omega_2}} P' \mid Q'}$$

(LCOMM2)
$$\frac{P \xrightarrow{(\nu a)(b,\mathbf{s})^i b_{\mathbf{s}}!l(a)} P' \quad Q \xrightarrow{(b,\mathbf{r})^j b_{\mathbf{r}}?l(a)} Q' \quad \omega = (b, \mathbf{s})^{1-i}(b, \mathbf{r})^{1-j} \quad a \notin \mathsf{fn}(Q)}{P \mid Q \xrightarrow{\tau_\omega} (\nu a)(P' \mid Q')}$$

(LAUTH1)
$$\frac{P \xrightarrow{(a,\mathbf{s})^i a_{\mathbf{s}}l\langle\mathbf{d}\rangle} P' \quad Q \xrightarrow{(a,\mathbf{r})^j a_{\mathbf{r}}l(\mathbf{d})} Q' \quad \mathbf{s} \neq \mathbf{d} \quad \omega = (a, \mathbf{s})^{1-i}(a, \mathbf{r})^{1-j} \quad i \in \{0, 1\} \quad j \in \{0, 1\}}{P \mid Q \xrightarrow{\tau_\omega^{(a,\mathbf{d})}} P' \mid (a, \mathbf{d})Q'}$$

(LAUTH2)
$$\frac{P \xrightarrow{a_{\mathbf{d}}l\langle\mathbf{d}\rangle} P' \quad Q \xrightarrow{(a,\mathbf{r})^i a_{\mathbf{r}}l(\mathbf{d})} Q' \quad \omega = (a, \mathbf{r})^{1-i} \quad i \in \{0, 1\}}{P \mid Q \xrightarrow{\tau_{(a,\mathbf{d})\omega}^{(a,\mathbf{d})}} P' \mid (a, \mathbf{d})Q'}$$

All four rules that describe invisible actions ((LCOMM1), (LCOMM2), (LAUTH1), (LAUTH2)) keep in ω and $\omega'$of $\tau_\omega^{\omega'}$ visible pairs that point to roles that were used in actions within corresponding invisible actions, but still require authorization.
Rules (LCOMM1) and (LCOMM2) are as in the π-calculus, with the main difference of having authorization pairs. If

$$\{\alpha_1, \alpha_2\} = \left\{(a, \mathbf{s})^i a_{\mathbf{s}}!l(b), (a, \mathbf{r})^j a_{\mathbf{r}}?l(b)\right\} \text{ or}$$
$$\{\alpha_1, \alpha_2\} = \left\{(a, \mathbf{s})^i(a, \mathbf{d})a_{\mathbf{s}}l\langle\mathbf{d}\rangle, (a, \mathbf{r})^j a_{\mathbf{r}}l(\mathbf{d})\right\},$$

where $i \in \{0, 1\}$, then we say that α_1 and α_2 are mutually dual actions and write $\alpha_1 = \overline{\alpha}_2$. If $i = 0$ in $(a, \mathbf{t})^i$ for some α, we write $\alpha_{(a,\mathbf{t})}$. For example:

$$b_{\mathbf{s}}!l(a).P' \mid b_{\mathbf{r}}?l(x).Q' \xrightarrow{\tau_{(b,\mathbf{s}),(b,\mathbf{r})}} P' \mid Q'\{a/x\},$$
$$(b, \mathbf{s})b_{\mathbf{s}}!l(a).P' \mid (b, \mathbf{r})b_{\mathbf{r}}?l(x).Q' \xrightarrow{\tau} P' \mid Q'\{a/x\},$$
$$(a, \mathbf{d})(a, \mathbf{d})a_{\mathbf{d}}l\langle\mathbf{d}\rangle.P' \mid (a, \mathbf{r})a_{\mathbf{r}}l(\mathbf{d}) \xrightarrow{\tau} (a, \mathbf{d})P' \mid (a, \mathbf{d})(a, \mathbf{r})Q';$$

Rules (LAUTH1) and (LAUTH2), characteristic of our calculus, describe an internal communication of an unauthorized role $\mathbf{d}$ along a name a between processes P and Q, continuing in parallel composition P and $(a, \mathbf{d})Q$. For example:

$$a_{\mathbf{s}} l\langle d\rangle.P' \mid a_{\mathbf{r}} l(d).Q' \xrightarrow{\tau^{(a,\mathbf{d})}_{(a,\mathbf{s}),(a,\mathbf{r})}} P' \mid (a, \mathbf{d})(a, \mathbf{d})Q', \tag{3}$$

$$(a, \mathbf{s})a_{\mathbf{s}} l\langle d\rangle.P' \mid a_{\mathbf{r}} l(d).Q' \xrightarrow{\tau^{(a,\mathbf{d})}_{(a,\mathbf{r})}} (a, \mathbf{s})P' \mid (a, \mathbf{d})(a, \mathbf{d})Q', \tag{4}$$

$$(a, \mathbf{s})a_{\mathbf{s}} l\langle d\rangle.P' \mid (a, \mathbf{r})a_{\mathbf{r}} l(d).Q' \xrightarrow{\tau^{(a,\mathbf{d})}} (a, \mathbf{s})P' \mid (a, \mathbf{d})(a, \mathbf{d})(a, \mathbf{r})Q', \tag{5}$$

(V) Finally, there are three rules (and three symmetric rules) that describe parallel composition contexts — Rules (LPAR1), (LPAR2), and (LPAR3)).

(LPAR1)
$$\frac{P \xrightarrow{\alpha} Q \qquad \mathsf{bn}(\alpha) \notin \mathsf{fn}(R) \quad \alpha \notin \{a_{\mathbf{s}} l\langle \mathbf{d}\rangle, (a, \mathbf{s})a_{\mathbf{s}} l\langle \mathbf{d}\rangle, \tau^{(a,\mathbf{d})}_{\omega}\}}{P \mid R \xrightarrow{\alpha} Q \mid R}$$

(LPAR2)
$$\frac{P \xrightarrow{(a,\mathbf{s})^i a_{\mathbf{s}} l\langle \mathbf{d}\rangle} Q \quad i \in \{0, 1\}}{P \mid R \xrightarrow{(a,\mathbf{s})^i a_{\mathbf{s}} l\langle \mathbf{d}\rangle} Q \mid (a, \mathbf{d})R}$$

(LPAR3)
$$\frac{P \xrightarrow{\tau^{(a,\mathbf{d})}_{\omega}} Q}{P \mid R \xrightarrow{\tau^{(a,\mathbf{d})}_{\omega}} Q \mid (a, \mathbf{d})R}$$

One should notice that not all transitions seamlessly cross parallel composition. By (LPAR2) and (LPAR3), in case a process can delegate not yet authorized role or communicate it within an internal action, then all processes in parallel get the authorization scope. For example:

$$(a, \mathbf{s})a_{\mathbf{s}} l\langle \mathbf{d}\rangle.P' \mid (a, \mathbf{r})a_{\mathbf{r}} l(\mathbf{d}).Q' \mid a_{\mathbf{d}}!l(b).R' \xrightarrow{\tau^{(a,\mathbf{d})}} (a, \mathbf{s})P' \mid (a, \mathbf{d})(a, \mathbf{d})(a, \mathbf{r})Q' \mid (a, \mathbf{d})a_{\mathbf{d}}!l(b).R'.$$

After closing the parallel composition with the same authorization scope, the process delegating the role will lose it and all others will keep it.

$$(a, \mathbf{d})((a, \mathbf{s})a_{\mathbf{s}} l\langle \mathbf{d}\rangle.P' \mid (a, \mathbf{r})a_{\mathbf{r}} l(\mathbf{d}).Q' \mid a_{\mathbf{d}}!l(b).R') \xrightarrow{\tau} (a, \mathbf{s})P' \mid (a, \mathbf{d})(a, \mathbf{d})(a, \mathbf{r})Q' \mid (a, \mathbf{d})a_{\mathbf{d}}!l(b).R'.$$

It is in a sense analogous to open and close rules for name scoping, but having in mind that we do not have authorization scope extrusion in the structural congruence (Table 2). Moreover, this treatment conveniently endows processes $(a, \mathbf{r})(P \mid Q)$ and $(a, \mathbf{r})P \mid (a, \mathbf{r})Q$ with the same behavior. For example,

$$(a, \mathbf{d})(a_{\mathbf{r}} l\langle \mathbf{d}\rangle.P \mid Q) \xrightarrow{(a,\mathbf{d})a_{\mathbf{r}} l\langle \mathbf{d}\rangle} P \mid (a, \mathbf{d})Q,$$
$$(a, \mathbf{d})(a_{\mathbf{r}} l\langle \mathbf{d}\rangle.P) \mid (a, \mathbf{d})Q \xrightarrow{(a,\mathbf{d})a_{\mathbf{r}} l\langle \mathbf{d}\rangle} P \mid (a, \mathbf{d})Q.$$

2.4. Properties

Having introduced reduction and LTS semantics for our process model, in what follows we prove their equivalence. We need some auxiliary results:

Lemma 1 *Let P and Q be processes and $i \in \{0, 1\}$.*

(i) *If* $P \xrightarrow{(b,\mathbf{s})^i b_{\mathbf{s}}!l(c)} Q$, *then* $P \equiv (\nu \vec{a}_1)(\delta(b, \mathbf{s})^i b_{\mathbf{s}}!l(c).P' \mid R)$ *and* $Q \equiv (\nu \vec{a}_1)(\delta(b, \mathbf{s})^i P' \mid R)$;

(ii) *If* $P \xrightarrow{(\nu c)(b,\mathbf{s})^i b_{\mathbf{s}}!l(c)} Q$, *then* $P \equiv (\nu \vec{a}_1)(\nu c)(\delta(b, \mathbf{s})^i b_{\mathbf{s}}!l(c).P' \mid R)$ *and* $Q \equiv (\nu \vec{a}_1)(\delta(b, \mathbf{s})^i P' \mid R)$;

(iii) *If* $P \xrightarrow{(b,\mathbf{s})^i (b,\mathbf{d})^i b_{\mathbf{s}} l\langle \mathbf{d}\rangle} Q$, *then* $P \equiv (\nu \vec{a}_1)(\delta(b, \mathbf{s})^i (b, \mathbf{d})^i b_{\mathbf{s}} l\langle d\rangle.P' \mid R)$ *and* $Q \equiv (\nu \vec{a}_1)(\delta(b, \mathbf{s})^i .P' \mid R)$;

(iv) *If* $P \xrightarrow{(b,\mathbf{r})^i b_{\mathbf{r}}?l(c)} Q$, *then* $P \equiv (\nu \vec{a}_1)(\delta(b, \mathbf{r})^i b_{\mathbf{r}}?l(x).P' \mid R)$ *and* $Q \equiv (\nu \vec{a}_1)(\delta(b, \mathbf{r})^i P'\{c/x\} \mid R)$;

(v) *If* $P \xrightarrow{(b,\mathbf{r})^i b_{\mathbf{r}} l(\mathbf{d})} Q$, *then* $P \equiv (\nu \vec{a}_1)(\delta(b,\mathbf{r})^i b_{\mathbf{r}} l(\mathbf{d}).P' \mid R)$ *and* $Q \equiv (\nu \vec{a}_1)(\delta(b,\mathbf{r})^i (b,\mathbf{d})P' \mid R)$;

(vi) *If* $P \xrightarrow{\tau^{(a,\mathbf{d})}} Q$ *then* $P \equiv (\nu \vec{a}_1)(\delta_1(a,\mathbf{s})a_{\mathbf{s}} l\langle d\rangle.P' \mid \delta_2(a,\mathbf{r})a_{\mathbf{r}} l(d).P' \mid R)$ *and* $(a,\mathbf{d}) \notin \delta_1$;

(vii) *If* $P \xrightarrow{\tau} Q$ *then either*

- $P \equiv (\nu \vec{a}_1)(\delta_1(b,\mathbf{d})(b,\mathbf{s})b_{\mathbf{s}} l\langle \mathbf{d}\rangle.P' \mid \delta_2(b,\mathbf{r})b_{\mathbf{r}} l(\mathbf{d}).P' \mid R)$ *and* $Q \equiv (\nu \vec{a}_1)(\delta_1(b,\mathbf{s}).P' \mid \delta_2(b,\mathbf{r})(b,\mathbf{d}).P' \mid R)$, *or*
- $P \equiv (\nu \vec{a}_1)(\delta_1(b,\mathbf{s})b_{\mathbf{s}}!l(c).P' \mid \delta_2(b,\mathbf{r})b_{\mathbf{r}}?l(x).P' \mid R)$ *and* $Q \equiv (\nu \vec{a}_1)(\delta_1(b,\mathbf{s}).P' \mid \delta_2(b,\mathbf{r})P'\{c/x\} \mid R)$.

Proof By induction on the length of the derivation of $P \xrightarrow{\alpha} Q$. □

Lemma 2 *If* $P \equiv P'$ *and* $P \xrightarrow{\alpha} Q$, *then there is* Q' *such that* $P' \xrightarrow{\alpha} Q'$ *and* $Q \equiv Q'$.

Proof By induction on the length of the derivation of $P \equiv P'$. □

Proposition 2 $P \to Q$ *if and only if there is* Q' *such that* $P \xrightarrow{\tau} Q'$ *and* $Q \equiv Q'$.

Proof The proof is in one direction by induction on derivation of $P \to Q$ and in opposite direction by induction on the derivation $P \xrightarrow{\tau} Q$. □

An advantage of defining an LTS is that it allows us to define behavioral equivalences. We say that a symmetric binary relation $\mathcal{B}$ on processes is a strong bisimulation if

$$(P, Q) \in \mathcal{B} \text{ and } P \xrightarrow{\alpha} P' \text{ and } \mathsf{bn}(\alpha) \cap \mathsf{fn}(Q) = \emptyset \text{ imply } Q \xrightarrow{\alpha} Q' \text{ and } (P', Q') \in \mathcal{B}, \text{ for some } Q'.$$

The largest strong bisimulation, the *strong bisimilarity*, is denoted by $\sim$.

Proposition 3 *If* $P \equiv Q$ *then* $P \sim Q$.

We may now state congruence of strong bisimilarity with respect to active contexts:

Proposition 4 *If* $P \sim Q$ *then* $\mathcal{C}[P] \sim \mathcal{C}[Q]$.

Proposition 4 attests that active contexts are proper functions of behavior: no two equivalent behaviors placed in an active context yield different (image) behaviors.

As part of the proof of Proposition 3 we prove $(a,\mathbf{r})(P \mid Q) \sim (a,\mathbf{r})P \mid (a,\mathbf{r})Q$ which serves as a behavioral validation of the authorization distribution law (cf. Table 2). We may also show that there is P such that $(a,\mathbf{r})(a,\mathbf{r})P \not\sim (a,\mathbf{r})P$ (consider P may delegate the authorization twice) so as to witness our authorization accounting principle. We may also show that $(a,\mathbf{r})\sigma.P \sim \sigma.(a,\mathbf{r})P$, for any P, provided that prefix σ does not involve channel a or role $\mathbf{r}$. In fact, since roles are statically prescribed we may show that $(a,\mathbf{r})P \sim P$ when role $\mathbf{r}$ does not occur in process P.

The behavioral equivalence, defined using the labeled transition system, serves as a proof method to show such behavioral (in)equalities, so as to further validate our design principles and inform on the behavioral model.

3. Type system

We now introduce a behavioral type system for our language. It ensures that well-typed processes are not (authorization) errors, in the sense of Definition 3 (Proposition 5). In turn, this property entails a natural form of type safety (Corollary 1).

Our type discipline builds on the conversation types language as presented in [BCVV12]: we extend message type M with the role $\mathbf{r}$, so that we may capture role authorization passing. This is a rather natural extension, formally given by the syntax in Table 4. Behavioral types B include:

- `end`, which describes inaction;
- $B \mid B$, which allows to describe concurrent independent behavior;

Table 4. Conversation types

B	::=	$\mathtt{end}$	(Inaction type)	M	::=	$B \mid T \mid \mathbf{r}$	(Message types)
	$\mid$	$B \mid B$	(Parallel type)	T	::=	$l(B)$	(Shared channel types)
	$\mid$	$\Diamond B$	(Sometime type)	p	::=	$!\mathbf{r} \mid ?\mathbf{r} \mid (\mathbf{r} \to \mathbf{r}):$	(Actions)
	$\mid$	$\mathtt{p}\, l(M).B$	(Action type)				

Table 5. Subtyping relation.

$$\frac{B_1 <: B_1'}{B_1 \mid B_2 <: B_1' \mid B_2} \qquad \frac{B <: B'}{\mathtt{p}l(M).B <: \mathtt{p}l(M).B'} \qquad \frac{\vdash \Diamond B}{B <: \Diamond B}$$

$$(B_1 \mid B_2) \mid B_3 \equiv B_1 \mid (B_2 \mid B_3) \qquad B_1 \mid B_2 \equiv B_2 \mid B_1 \qquad B \mid \mathbf{end} \equiv B$$

$$\Diamond(B_1 \mid B_2) \equiv \Diamond B_1 \mid \Diamond B_2 \qquad \Diamond\mathbf{end} \equiv \mathbf{end}$$

– the sometime type $\Diamond B$, which says that behavior B may take place immediately or later on;
– $\mathtt{p}l(M).B$, which describes a communication action identifying the role(s) involved, and whether the action is an input $?\mathbf{r}$ or an output $!\mathbf{r}$ or a message exchange $(\mathbf{r} \to \mathbf{r})$:, a carried message type M and the behavior B that is prescribed to take place after the communication action.

We use behavioral types to specify the interactions in *linear* channels, where no communication races are allowed (which is to say that at any given moment, there can only be one matching pair of input/output actions). In our setting, where several parties may simultaneously use a channel, this *linear* communication pairing is ensured via message tags: at a given moment, there can be only one pair of processes able to exchange a tagged message. For shared channels, where communication races are allowed, we ensure consistent usage (but no structured protocol of interaction) via shared channel types T, which carry a (linear) behavioral type describing the usage delegated in the communication.

Message type M also captures the usages delegated in communications: in case M is a behavioral type B or a shared channel type T it describes how the receiving process uses the received channel; in case M identifies a role $\mathbf{r}$ then the message type captures an authorization delegation in the specified role.

Type environments separate linear and shared channel usages: Δ associates channels with (linear) behavioral types, given by $\Delta ::= \emptyset \mid \Delta, a : B$, whereas Γ associates channels with shared channel types, given by $\Gamma ::= \emptyset \mid \Gamma, a : T$.

Typing rules rely on *subtyping* as well as on some auxiliary notions of *apartness*, *wellformedness*, and *splitting* of types. Explanations for these notions are in order:

– The subtyping relation is given in Table 5, where we use $B_1 \equiv B_2$ iff $B_1 <: B_2$ and $B_2 <: B_1$. Apart from some usual structural identifications and context closure, subtyping specifies that $B <: \Diamond B$ (provided the latter is well-formed) which allows to say that a behavior B that is prescribed to take place immediately may as well be used in a context that expects B to take place later on in the protocol.
– Type apartness is used to identify non-interfering concurrent behaviors that may be safely composed in a linear interaction. To define type-apartness we use $tg(B)$ to denote the set of tags occurring in type B, defined as expected. Two types B_1 and B_2 are apart, written $B_1 \# B_2$, if they have disjoint sets of message tags ($tg(B_1) \cap tg(B_2) = \emptyset$).
– Well-formedness of a type (Table 6) ensures that parallel behaviors are apart and that the sometime $\Diamond$ is not associated to message synchronizations — synchronizations are not allowed to take place sometime later, they are always specified to take place at a given stage in the protocol. Also, well-formedness ensures that parallel behaviors are independent (via apartness # that checks disjointness of message tags sets).

Table 6. Well-formed type predicate

$\vdash \mathbf{end}$	$\dfrac{B_1 \# B_2 \quad \vdash B_1 \quad \vdash B_2}{\vdash B_1 \mid B_2}$	$\dfrac{\vdash B \quad \mathtt{p}\,l(M).\mathbf{end} \# B}{\vdash \mathtt{p}\,l(M).B}$
$\vdash \Diamond\mathbf{end}$	$\dfrac{\vdash B_1 \mid B_2 \quad \vdash \Diamond B_1 \quad \vdash \Diamond B_2}{\vdash \Diamond(B_1 \mid B_2)}$	$\dfrac{\vdash \mathtt{p}\,l(M).B \quad \mathtt{p} \in \{!\mathbf{s}, ?\mathbf{r}\}}{\vdash \Diamond \mathtt{p}\,l(M).B}$

Table 7. Type Split Relation

$\dfrac{\vdash B}{B = \mathbf{end} \circ B} \qquad \dfrac{B_1 = B_1' \circ B_1'' \quad B_2 = B_2' \circ B_2'' \quad \vdash B_1 \mid B_2}{B_1 \mid B_2 = B_1' \mid B_2' \circ B_1'' \mid B_2''}$	(S-END,S-PAR)
$\dfrac{B = B_1 \circ B_2 \quad \{\mathtt{p}_1, \mathtt{p}_2\} = \{!\mathbf{r}_1, ?\mathbf{r}_2\} \quad \vdash (\mathbf{r}_1 \to \mathbf{r}_2) : l(M).B}{(\mathbf{r}_1 \to \mathbf{r}_2) : l(M).B = \mathtt{p}_1\, l(M).B_1 \circ \Diamond \mathtt{p}_2\, l(M).B_2}$	(S-TAU)
$\dfrac{B_1 = B_2 \circ \Diamond B \quad \vdash \mathtt{p}\,l(M).B_1 \quad \mathtt{p}\,l(M).\mathbf{end} \# \Diamond B}{\mathtt{p}\,l(M).B_1 = \mathtt{p}\,l(M).B_2 \circ \Diamond B}$	(S-BRK)
$\dfrac{B_1 = B_2 \circ \Diamond B \quad \vdash \Diamond \mathtt{p}\,l(M).B_1 \quad \Diamond \mathtt{p}\,l(M).\mathbf{end} \# \Diamond B}{\Diamond \mathtt{p}\,l(M).B_1 = \Diamond \mathtt{p}\,l(M).B_2 \circ \Diamond B}$	(S-BRKS)
$\dfrac{B = B_2 \circ B_1}{B = B_1 \circ B_2} \qquad \dfrac{B_1' = B_2' \circ B_3' \quad B_1 \equiv B_1' \quad B_2 \equiv B_2' \quad B_3 \equiv B_3'}{B_1 = B_2 \circ B_3}$	(S-SYM,S-EQU)

Table 8. Reduction of B and Δ.

$(\mathbf{s} \to \mathbf{r}) : l(M).B \to B$	$\dfrac{B_1 \to B_2}{B_1 \mid B \to B_2 \mid B}$	$\dfrac{B_1 \to B_2}{B \mid B_1 \to B \mid B_2}$	$\emptyset \to \emptyset$	$\dfrac{\Delta \to \Delta'}{\Delta, a : B \to \Delta', a : B}$	$\dfrac{B_1 \to B_2}{\Delta, a : B_1 \to \Delta, a : B_2}$

– Type splitting given in Table 7 supports the distribution of protocol "slices" among the participants in a conversation: we write $B = B_1 \circ B_2$ to say that behavior B may be split in behaviors B_1 and B_2 so that an overall behavior B may be distributed (e.g., in the two branches of a parallel composition). We remark that B_1 and B_2 may be further split so as to single out the individual contributions of each participant in a conversation, where decomposition is driven by the structure of the process in the typing rules. We highlight Rule (S-TAU) that allows to distribute a synchronization in the two (dual) communication prefixes, where one of them is prescribed to happen sometime ($\Diamond$) while the other is prescribed to take place immediately (so as to ensure the synchronization takes place immediately). Rule (S-BRK) allows to extract a ($\Diamond$) part of the protocol that is prescribed to happen after a prefix (provided the extracted behavior and the prefix are independent) so that, for instance, an already active third party that will only be engaged later on in the protocol may be initially characterized by the respective sometime behavior extracted from the later stages of the protocol.
We lift the split relation to Δ type environments in unsurprising lines: $\Delta, a : B = \Delta_1, a : B_1 \circ \Delta_2, a : B_2$ if $B = B_1 \circ B_2$ and $\Delta = \Delta_1 \circ \Delta_2$, and also $\Delta, a : B = \Delta_1, a : B \circ \Delta_2$ (and symmetrically) if $\Delta = \Delta_1 \circ \Delta_2$. In typing rules we write $\Delta_1 \circ \Delta_2$ to represent Δ (if there is such Δ) such that $\Delta = \Delta_1 \circ \Delta_2$.

Moreover, behavioral type and linear type environment reduction are given in Table 8, where the former states that a synchronized communication prefix can reduce to its continuation, including under parallel composition, while the latter lifts the relation to environments by allowing any type in the environment to reduce, embedding reflexivity via $\emptyset \to \emptyset$.

Table 9. Typing rules

(T-END) $\dfrac{}{\Delta_{\mathtt{end}};\ \Gamma \vdash_{\emptyset} 0}$ (T-AUTH) $\dfrac{\Delta;\ \Gamma \vdash_{\Sigma} P \quad \Xi = \Sigma \setminus \{(a, \mathbf{r})\}}{\Delta;\ \Gamma \vdash_{\Xi} (a, \mathbf{r})P}$ (T-PAR) $\dfrac{\Delta_1;\ \Gamma \vdash_{\Sigma} P \qquad \Delta_2;\ \Gamma \vdash_{\Xi} Q}{\Delta_1 \circ \Delta_2;\ \Gamma \vdash_{\Sigma \cup \Xi} P \mid Q}$

(T-SNEW) $\dfrac{\Delta;\ \Gamma, a : l(B) \vdash_{\Sigma} P}{\Delta;\ \Gamma \vdash_{\Sigma} (\nu a)P}$ (T-NEW) $\dfrac{\Delta, a : B;\ \Gamma \vdash_{\Sigma} P \qquad \mathtt{matched}(B) \qquad a \notin \mathtt{pr}_1(\Sigma)}{\Delta;\ \Gamma \vdash_{\Sigma} (\nu a)P}$

(T-ROLEIN) $\dfrac{\Delta \circ a : B;\ \Gamma \vdash_{\Sigma} P \quad ?\mathbf{r}\, l(\mathbf{s}).B <: B' \quad \Xi = (\Sigma \setminus \{(a, \mathbf{s})\}) \cup \{(a, \mathbf{r})\}}{\Delta \circ a : B';\ \Gamma \vdash_{\Xi} a_{\mathbf{r}}\, l(\mathbf{s}).P}$

(T-ROLEOUT) $\dfrac{\Delta \circ a : B;\ \Gamma \vdash_{\Sigma} P \quad !\mathbf{r}\, l(\mathbf{s}).B <: B' \quad (a, \mathbf{s}) \notin \Sigma \quad \Xi = \Sigma \cup \{(a, \mathbf{r}), (a, \mathbf{s})\}}{\Delta \circ a : B';\ \Gamma \vdash_{\Xi} a_{\mathbf{r}}\, l\langle \mathbf{s} \rangle.P}$

(T-IN) $\dfrac{\Delta \circ a : B, b : B';\ \Gamma \vdash_{\Sigma} P \quad ?\mathbf{r}\, l(B').B <: B'' \quad b \notin \mathtt{pr}_1(\Sigma) \quad \Xi = \Sigma \cup \{(a, \mathbf{r})\}}{\Delta \circ a : B'';\ \Gamma \vdash_{\Xi} a_{\mathbf{r}} ?l(b).P}$

(T-OUT) $\dfrac{\Delta \circ a : B;\ \Gamma \vdash_{\Sigma} P \qquad !\mathbf{r}\, l(B').B <: B'' \qquad \Xi = \Sigma \cup \{(a, \mathbf{r})\}}{\Delta \circ a : B'' \circ b : B';\ \Gamma \vdash_{\Xi} a_{\mathbf{r}} !l(b).P}$

(T-LSIN) $\dfrac{\Delta \circ a : B;\ \Gamma, b : T \vdash_{\Sigma} P \quad ?\mathbf{r}\, l(T).B <: B' \quad b \notin \mathtt{pr}_1(\Sigma) \quad \Xi = \Sigma \cup \{(a, \mathbf{r})\}}{\Delta \circ a : B';\ \Gamma \vdash_{\Xi} a_{\mathbf{r}} ?l(b).P}$

(T-LSOUT) $\dfrac{\Delta \circ a : B;\ \Gamma, b : T \vdash_{\Sigma} P \qquad !\mathbf{r}\, l(T).B <: B' \qquad \Xi = \Sigma \cup \{(a, \mathbf{r})\}}{\Delta \circ a : B';\ \Gamma, b : T \vdash_{\Xi} a_{\mathbf{r}} !l(b).P}$

(T-SIN) $\dfrac{\Delta, b : B;\ \Gamma, a : l(B) \vdash_{\Sigma} P \quad b \notin \mathtt{pr}_1(\Sigma) \quad \Xi = \Sigma \cup \{(a, \mathbf{r})\}}{\Delta;\ \Gamma, a : l(B) \vdash_{\Xi} a_{\mathbf{r}} ?l(b).P}$

(T-SOUT) $\dfrac{\Delta;\ \Gamma, a : l(B) \vdash_{\Sigma} P \quad \Xi = \Sigma \cup \{(a, \mathbf{r})\}}{\Delta \circ b : B;\ \Gamma, a : l(B) \vdash_{\Xi} a_{\mathbf{r}} !l(b).P}$

A typing judgment is of the form $\Delta;\ \Gamma \vdash_{\Sigma} P$, where Σ denotes the *authorization set*. This is a subset of the direct product of the set of channel names and the set of roles, i.e., $\Sigma \subseteq \mathcal{N} \times \mathcal{R}$. The typing judgment $\Delta;\ \Gamma \vdash_{\Sigma} P$ states that the process P is well typed under Δ and Γ with roles from $\mathtt{pr}_2(\Sigma)$ (the projection on the second element of the pairs in Σ) appearing in P unauthorized on corresponding channels from $\mathtt{pr}_1(\Sigma)$.

Typing rules are presented in Fig. 9. There are three rules that are specific to our model:

- (T-AUTH) types a process under authorization scope $(a, \mathbf{r})$ with authorization set diminished by $(a, \mathbf{r})$.
- (T-ROLEIN) types authorization reception of the role $\mathbf{s}$ on the linear channel a, under the role $\mathbf{r}$, with the authorization set diminished by $(a, \mathbf{s})$, and enlarged with $(a, \mathbf{r})$.
- (T-ROLEOUT) types sending of the authorization $\mathbf{s}$ on linear channel a, under the role $\mathbf{r}$, with the authorization set enlarged with $(a, \mathbf{r})$ and $(a, \mathbf{s})$.

Notice that in Rules (T-ROLEIN) and (T-ROLEOUT) the typing environment in the conclusion is split in a typing of a that specifies the reception of the authorization (up to subtyping).

All other rules are similar to the typing rules from [BCVV12], with the derivation of the novel decoration Σ as follows. Rule (T-END) states that a well-typed inactive process has no unauthorized roles and only `end` usages of linear channels (denoted by $\Delta_{\mathtt{end}}$). Rule (T-NEW) types a restricted linear name if it (a) has no unauthorized roles and (b) has no un*matched* communications (no output or input communication prefixes). Rule (T-SNEW) types a restricted shared name, without any additional restriction on unauthorized roles. Rules (T-IN), (T-OUT), (T-LSIN), (T-LSOUT), (T-SIN) and (T-SOUT) type input and output actions under the role $\mathbf{r}$, with the authorization set enlarged with $(a, \mathbf{r})$. The authorization is not performed on shared channels, implying that all actions on shared channels in a typed process must be under the corresponding authorization scope. Rule (T-PAR) states that the unauthorized pairs in a parallel composition of two processes is the union of unauthorized pairs of the two composed processes. We say that a process P is *well typed* if there are Δ and Γ such that $\Delta;\ \Gamma \vdash_{\emptyset} P$.

Example **4** Consider process $OtherProvider$ from our motivating example (§1.3). Using Rules (T-AUTH), (T-IN), (T-ROLEIN), and (T-OUT) given in the Table 9 the reader can verify that process $OtherProvider$ is well typed, written

$$\Delta;\ \emptyset \vdash_{\emptyset} OtherProvider$$

where

$$OtherProvider \triangleq (service, \mathbf{slave})service_{\mathbf{slave}}?delegate(y).(y, \mathbf{slave})y_{\mathbf{slave}}auth(\mathbf{server}).y_{\mathbf{server}}!finalize()$$

and

$$\begin{array}{ll}\Delta & = service :?\mathbf{slave}\, delegate(B_y).\mathtt{end} \ \circ\ y : B_y\\ \text{with}\\ B_y & = ?\mathbf{slave}\, auth(\mathbf{server}).!\mathbf{server}\ finalize().\mathtt{end}.\end{array}$$

Contrary, the process

$$EvilProvider \triangleq (service, \mathbf{slave})service_{\mathbf{slave}}?delegate(y).y_{\mathbf{server}}!evilthing().$$

that attempts using the channel y under role **server** without receiving the authorization for this role, is not typable.

We define the reduction relation $\rightarrow$ between behavioral types B and corresponding environments Δ by allowing a synchronized communication prefix to reduce to its continuation $(\mathbf{s} \rightarrow \mathbf{r}) : l(M).B \rightarrow B$, so as to mimic the respective process behavior, by allowing reduction to occur in a branch of a parallel composition (e.g., $B_1 \rightarrow B_2 \Rightarrow B \mid B_1 \rightarrow B \mid B_2$), and by lifting the relation point-wise to environments, embedding reflexivity so as to encompass process reductions involving shared or bound channels (where no reduction in the linear usages of free names is required).

Theorem 1 (Type preservation) *Let* $\Delta;\ \Gamma \vdash_{\Sigma} P$ *for some* Δ, Γ, Σ *and* P. *If* $P \rightarrow Q$ *then there is* Δ' *such that* $\Delta \rightarrow \Delta'$ *and* $\Delta';\ \Gamma \vdash_{\Sigma} Q$.

A direct consequence of type preservation is *protocol fidelity*: every reduction of the process corresponds to a reduction of the types, thus ensuring that the process follows the protocols prescribed by the types. Notice that communication safety (no type mismatches in communications) is entailed by protocol fidelity, which in our case also attests that processes agree in the role when sending and receiving authorizations.

Proposition 5 (Error free) *If* P *is a well-typed process, then* P *is not an authorization error.*

Combining freedom from errors and type preservation results we immediately obtain our notion of type safety, which ensures that well-typed processes never reach an error configuration.

Corollary 1 (Type safety) *If* P *is a well-typed process and* $P \rightarrow^* Q$, *then* Q *is not an authorization error.*

Our type safety result ensures every action that may eventually become active is duly authorized. Notice that our typing characterization directly records the authorizations required by the system from the external context. However, we may also infer the authorizations held by the system itself, by contrasting the channel usage

(captured by the typing environment) with the required authorizations: channels that are not mentioned in the required authorizations that have a prescribed usage must be authorized in the system. As such the behavioral interface captured by the typing characterization can also be viewed as a system specification: provided some authorizations, the system behavior must be compliant with the prescribed interface, possibly including actions for which authorization is not provided.

4. Related work

From a broad, historical perspective, studies on access control and authorization go back at least to the seminal work by Lampson [Lam74], who proposed the *access control matrix model*. Harrison, Ruzzos and Ullman (HRU) [HRU76] formalized this protection model as a matrix which has columns indexed by objects and rows indexed by subjects; each entry in the matrix is a subject of a set of access models. HRU proved that it is undecidable whether a given state of a given protection system is safe for a given generic right. Sandhu [San92] developed the *typed* access matrix (TAM) model by introducing strong typing into the HRU model (in essence, by assigning types to objects), and showed that strong typing usefully enables decidable safety analyses. See [SdV00] for an introduction to access control systems and their formalization.

Fournet et al. [FGM07] consider the problem of verifying whether a distributed system correctly implements a target authorization policy, expressed as *statements* and *expectations*. Their formal development is based on the spi-calculus, a variant of the π-calculus with cryptographic operations. They develop a type system based on annotations that define authorization policies: the type system checks conformance to a policy by tracking the logical facts/rules in the typing environment, and using logical deduction to type authorization expectations.

Role-based access control in distributed systems with dynamic access rights was handled by Dezani-Ciancaglini et al. [DGJP10] by means of a type system, which ensures security properties. In this calculus, roles assigned to data can be dynamically administered, while role communication between processes was not treated. Previous works in the setting of session calculi consider security properties like confidentiality and integrity (we refer the interested reader to the recent overview [BCD$^+$15]). For instance, Bonelli et al. [BCG05] extend session types with *correspondence assertions*, a form of dependent types which ensures consistency of data during computation. More recently, aspects of secure information flow and access control have been addressed for sessions in works such as [BCCDC11, CCDC11, CCDCR10]. Deniélou and Yoshida [DY11] use a kind of role-based approach where communication is controlled by a previously acquired reputation. Similarly to our work, Lapadula et al. [LPT07] consider a typed approach to role-based authorizations for service-oriented applications. Differently from our model, assigned roles are initially authorized and communicated data carries information on roles that will use it.

Scoping operators have been used to model security aspects; they typically rely on bound names to represent secrets or nonces. With the aim of representing secrecy and confidentiality requirements in process specifications, Giunti et al. [GPV12] investigate an scoping operator called *hide*, alternative to usual name restriction; the hide operator is embedded in the so-called *secrecy* π-calculus, tailored to program secrecy in communications. The expressiveness of the hide operator is investigated in the context of a behavioral theory, by means of a Spy agent. In contrast, our (free name) scoping operator focuses on authorization, a different security concern.

Vivas and Yoshida [VY02] propose a scoping operator (called *filter*) for dynamic channel screening. In a different setting (higher-order process communication) and with similar properties, the filter operator blocks all the actions that are not contained in the corresponding filter (which contains polarized channel names). Contrary to our authorization scope construct, filters are statically assigned to processes, while the authorization scope assigned to a process may be dynamically changed.

To the best of our knowledge, the authorization scoping proposed here has not been explored before for the specification of communication-centered systems. Our integration of the authorization scoping in the system specification language in an uniform way is a distinguished feature of our approach when compared to other forms of site-level authorization handling (see, e.g., Gorla and Pugliese's work [GP09]). There are high-level similarities between our work and the concept of *ownership types*, as well studied for object-oriented languages [CPN98]. Although in principle ownership types focus on static ownership structures, assessing their use for disciplining dynamic authorizations is interesting future work. Also, it may also prove to be worthwhile to exploit dependent/refinement types (see, e.g., [SCC10]) in our setting.

5. Concluding remarks

In this paper, we have introduced a typed process framework for specifying and reasoning about role-based authorization. Our contribution is based on previous work on conversation types [CV10] and their extension with dynamic assignment of roles to several parties in a concurrent system [BCVV12]. We focused on a modular extension of the existing framework, so as to leverage on previous results, adding the minimal elements so as to identify the specific issues at hand and set the basis for further developments.

We have addressed the problem of role authorization and communication of authorizations in an extension of the π-calculus, where communication prefixes are annotated with role authorizations. For simplicity, we have considered a "flat" set of roles; it is conceivable that concrete application settings may require sets of roles subject to hierarchical orderings. The presented calculus allows for the dynamic communication of authorizations and is equipped with an authorization scoping operator that precisely specifies the range of authorized use of particular roles in particular channels. We then extend the conversation type system in which a well-typed process can never reduce to an authorization error, thus (statically) ensuring that well-typed processes are always authorized to communicate on behalf of a role, including when authorizations are dynamically passed in messages.

The analysis technique presented in this paper combines a behavioral type discipline with authorization control. In our view, structured communication and authorization are orthogonal aspects subject to be controlled in separate, dedicated ways. Our type structure retains the meaning of types in [CV10, BCVV12] and does not mention authorizations, which are instead treated at the level of process specifications. This separation between communication and authorizations leads to clear typed interfaces and simplifies analyses. Still, we may already argue that our behavioral types pave the way for the verification of more fine-grained authorization communication control. Indeed, our types mention explicitly the identities of the communicated roles; this allows us to enforce further discipline just by inspecting the types. Consider, for example, the type

$$?\mathbf{r}\, l(\mathbf{d}).(\mathbf{s} \to \mathbf{t}) : l(\mathbf{d}).\mathtt{end}$$

where the specification yields the information that role $\mathbf{d}$ is received and afterwards communicated away. When role authorization containment is a concern, for instance since trust can be of an intransitive nature, it may be desirable to exclude the above communication pattern under some circumstances. One possibility is to completely exclude authorization re-communication. Alternatively we can look at the involved roles and validate according to some trust policy defined for them.

The advantage of lifting the analysis from processes to types is that the simpler the target of the analysis is, the simpler the analysis techniques will be. Such a verification step can be carried out statically, when nevertheless also the system must be verified, but more interestingly, can also be carried out at runtime. If we interpret such types as contracts to which processes will abide to, as ensured by the fidelity property, the composition of systems at runtime may be driven by type information. For instance, one may choose to avoid using web apps that re-communicate authorizations under the chosen circumstances. Investigating this kind of static analysis over types is an interesting direction for the future.

The type structure presented in this paper is purposefully simple and we do not consider constructs for infinite behavior at the level of processes nor at the level of behavioral types. Indeed, the model and examples considered throughout the paper are finite-state, and therefore possibly amenable for analysis using alternative techniques, such as, e.g., reachability analysis/model checking. In the presence of infinite behavior and recursive types, the situation is different. Although recursive types are absent in [BCVV12], we find no fundamental obstacle in extending our model with recursive types, essentially following the approach in [CV10] for conversation types. We believe that recursion and authorization handling are orthogonal concerns; further work is required to clarify this relationship.

Acknowledgements

We thank the anonymous referees for their insightful remarks that allowed us to greatly improve the presentation of our work. We acknowledge support by COST Action IC1201: Behavioural Types for Reliable Large-Scale Software Systems (BETTY) via Short-Term Scientific Mission grants (to Pantović and Vieira), and by grants ON174026 and III44006 of the Ministry of Education and Science, Serbia. Pérez is also affiliated to the NOVA Laboratory for Computer Science and Informatics (NOVA LINCS – PEst/UID/CEC/04516/2013), Universidade Nova de Lisboa, Portugal.

A. Proofs

A.1. Proofs from Sect. 2

Lemma 1 *Let P and Q be processes and $i \in \{0, 1\}$.*

(i) *If* $P \xrightarrow{(b,\mathbf{s})^i b_{\mathbf{s}}!l(c)} Q$, *then* $P \equiv (\nu \vec{a}_1)(\delta(b, \mathbf{s})^i b_{\mathbf{s}}!l(c).P' \mid R)$ *and* $Q \equiv (\nu \vec{a}_1)(\delta(b, \mathbf{s})^i P' \mid R)$;

(ii) *If* $P \xrightarrow{(\nu c)(b,\mathbf{s})^i b_{\mathbf{s}}!l(c)} Q$, *then* $P \equiv (\nu \vec{a}_1)(\nu c)(\delta(b, \mathbf{s})^i b_{\mathbf{s}}!l(c).P' \mid R)$ *and* $Q \equiv (\nu \vec{a}_1)(\delta(b, \mathbf{s})^i P' \mid R)$;

(iii) *If* $P \xrightarrow{(b,\mathbf{s})^i (b,\mathbf{d})^i b_{\mathbf{s}} l\langle \mathbf{d} \rangle} Q$, *then* $P \equiv (\nu \vec{a}_1)(\delta(b, \mathbf{s})^i (b, \mathbf{d})^i b_{\mathbf{s}} l\langle d \rangle.P' \mid R)$ *and* $Q \equiv (\nu \vec{a}_1)(\delta(b, \mathbf{s})^i .P' \mid R)$;

(iv) *If* $P \xrightarrow{(b,\mathbf{r})^i b_{\mathbf{r}}?l(c)} Q$, *then* $P \equiv (\nu \vec{a}_1)(\delta(b, \mathbf{r})^i b_{\mathbf{r}}?l(x).P' \mid R)$ *and* $Q \equiv (\nu \vec{a}_1)(\delta(b, \mathbf{r})^i P'\{c/x\} \mid R)$;

(v) *If* $P \xrightarrow{(b,\mathbf{r})^i b_{\mathbf{r}} l(\mathbf{d})} Q$, *then* $P \equiv (\nu \vec{a}_1)(\delta(b, \mathbf{r})^i b_{\mathbf{r}} l(\mathbf{d}).P' \mid R)$ *and* $Q \equiv (\nu \vec{a}_1)(\delta(b, \mathbf{r})^i (b, \mathbf{d})P' \mid R)$;

(vi) *If* $P \xrightarrow{\tau^{(a,\mathbf{d})}} Q$ *then* $P \equiv (\nu \vec{a}_1)(\delta_1(a, \mathbf{s})a_{\mathbf{s}} l\langle d \rangle.P' \mid \delta_2(a, \mathbf{r})a_{\mathbf{r}} l(d).P' \mid R)$ *and* $(a, \mathbf{d}) \notin \delta_1$;

(vii) *If* $P \xrightarrow{\tau} Q$ *then either*

- $P \equiv (\nu \vec{a}_1)(\delta_1(b, \mathbf{d})(b, \mathbf{s})b_{\mathbf{s}} l\langle \mathbf{d} \rangle.P' \mid \delta_2(b, \mathbf{r})b_{\mathbf{r}} l(\mathbf{d}).P' \mid R)$ *and* $Q \equiv (\nu \vec{a}_1)(\delta_1(b, \mathbf{s}).P' \mid \delta_2(b, \mathbf{r})(b, \mathbf{d}).P' \mid R)$, *or*
- $P \equiv (\nu \vec{a}_1)(\delta_1(b, \mathbf{s})b_{\mathbf{s}}!l(c).P' \mid \delta_2(b, \mathbf{r})b_{\mathbf{r}}?l(x).P' \mid R)$ *and* $Q \equiv (\nu \vec{a}_1)(\delta_1(b, \mathbf{s}).P' \mid \delta_2(b, \mathbf{r})P'\{c/x\} \mid R)$.

Proof The proof is by rule induction on the inference of $P \xrightarrow{\alpha} Q$. We will highlight the following case:

(ii) Base case is obtained by rule (LRES2). In that case, $P = (\nu c)P_1$ and $P_1 \xrightarrow{(b,\mathbf{s})^i b_{\mathbf{s}}!l(c)} Q$. By (i) we have

$$P_1 \equiv (\nu \vec{a}_1)(\delta(b, \mathbf{s})^i b_{\mathbf{s}}!l(c).P' \mid R) \quad \text{and} \quad Q \equiv (\nu \vec{a}_1)(\delta(b, \mathbf{s})^i P' \mid R).$$

and therefore, applying (SC-REST-COMM), we have

$$P \equiv (\nu c)(\nu \vec{a}_1)(\delta(b, \mathbf{s})^i b_{\mathbf{s}}!l(c).P' \mid R) \equiv (\nu \vec{a}_1)(\delta(b, \mathbf{s})^i b_{\mathbf{s}}!l(c).P' \mid R).$$

Induction step considers the following cases of the last applied rule:

1. (LRES1) : $P = (\nu a)P_1$ and $Q = (\nu a)Q_1$ and $P_1 \xrightarrow{(\nu c)(b,\mathbf{s})^i b_{\mathbf{s}}!l(c)} Q_1$ and $a \notin \{b, c\}$. By induction hypothesis,

$$P_1 \equiv (\nu \vec{a}_1)(\nu c)(\delta(b, \mathbf{s})^i b_{\mathbf{s}}!l(c).P' \mid R) \quad \text{and} \quad Q_1 \equiv (\nu \vec{a}_1)(\delta(b, \mathbf{s})^i P' \mid R)$$

and therefore

$$P \equiv (\nu a)(\nu \vec{a}_1)(\nu c)(\delta(b, \mathbf{s})^i b_{\mathbf{s}}!l(c).P' \mid R) \quad \text{and} \quad Q \equiv (\nu a)(\nu \vec{a}_1)(\delta(b, \mathbf{s})^i P' \mid R).$$

2. (LSCOPE1):

(a) $P = (a, \mathbf{t})P_1$ and $Q = (a, \mathbf{t})Q_1$ and $(a, \mathbf{t}) \neq (b, \mathbf{s})$ and $P_1 \xrightarrow{(\nu c)(b,\mathbf{s})^i b_{\mathbf{s}}!l(c)} Q_1$. By induction hypothesis

$$P_1 \equiv (\nu \vec{a}_1)(\nu c)(\delta(b, \mathbf{s})^i b_{\mathbf{s}}!l(c).P' \mid R) \qquad Q_1 \equiv (\nu \vec{a}_1)(\delta(b, \mathbf{s})^i P' \mid R)$$

and, by (SC-AUTH-SCOPE), assuming $a \neq c$, $a \notin \vec{a}_1$, and (SC-AUTH-DIST), we get

$$\begin{aligned} P &\equiv (a, \mathbf{t})(\nu \vec{a}_1)(\nu c)(\delta(b, \mathbf{s})^i b_{\mathbf{s}}!l(c).P' \mid R) \equiv (\nu \vec{a}_1)(\nu c)(a, \mathbf{t})(\delta(b, \mathbf{s})^i b_{\mathbf{s}}!l(c).P' \mid R) \\ &\equiv (\nu \vec{a}_1)(\nu c)((a, \mathbf{t})\delta(b, \mathbf{s})^i b_{\mathbf{s}}!l(c).P' \mid (a, \mathbf{t})R) \text{ and} \\ Q &\equiv (a, \mathbf{t})(\nu \vec{a}_1)(\delta(b, \mathbf{s})^i P' \mid R) \equiv (\nu \vec{a}_1)((a, \mathbf{t})\delta(b, \mathbf{s})^i P' \mid (a, \mathbf{t})R). \end{aligned}$$

(b) $P = (b, \mathbf{s})P_1$ and $Q = (b, \mathbf{s})Q_1$ and $i = 1$ and $P_1 \xrightarrow{(\nu c)b_{\mathbf{s}}!l(c)} Q_1$. By induction hypothesis

$$P_1 \equiv (\nu \vec{a}_1)(\nu c)(\delta b_{\mathbf{s}}!l(c).P' \mid R) \quad \text{and} \quad Q_1 \equiv (\nu \vec{a}_1)(\delta P' \mid R).$$

By (SC-AUTH-SCOPE), $b \neq c$, $b \notin \vec{a}_1$, and (SC-AUTH-DIST), we get

$$P \equiv (b, \mathbf{s})(\nu \vec{a}_1)(\nu c)(\delta b_{\mathbf{s}}!l(c).P' \mid R) \equiv (\nu \vec{a}_1)(\nu c)((b, \mathbf{s})\delta b_{\mathbf{s}}!l(c).P' \mid (b, \mathbf{s})R) \text{ and}$$
$$Q \equiv (b, \mathbf{s})(\nu \vec{a}_1)(\delta P' \mid R) \equiv (\nu \vec{a}_1)((b, \mathbf{s})\delta P' \mid (b, \mathbf{s})R).$$

3. (LPAR1) : $P = P_1 \mid R$ and $Q = Q_1 \mid R$ and $P_1 \xrightarrow{(\nu c)(b,\mathbf{s})^i b_{\mathbf{s}}!l(c)} Q_1$ and $c \notin \mathsf{fn}(R)$. By induction hypothesis

$$P_1 \equiv (\nu \vec{a}_1)(\nu c)(\delta(b, \mathbf{s})^i b_{\mathbf{s}}!l(c).P' \mid R_1) \quad \text{and } Q_1 \equiv (\nu \vec{a}_1)(\delta(b, \mathbf{s})^i P' \mid R_1).$$

By (SC-REST-EXTR) and (SC-ALPHA), for $\vec{a}_1$, $c \not\sqsubseteq \mathsf{fn}(R)$, we get

$$P \equiv R \mid (\nu \vec{a}_1)(\nu c)(\delta(b, \mathbf{s})^i b_{\mathbf{s}}!l(c).P' \mid R_1) \equiv (\nu \vec{a}_1)(\nu c)(\delta(b, \mathbf{s})^i b_{\mathbf{s}}!l(c).P' \mid R_1 \mid R) \text{ and}$$
$$Q \equiv R \mid (\nu \vec{a}_1)(\delta(b, \mathbf{s})^i P' \mid R_1) \equiv (\nu \vec{a}_1)(\delta(b, \mathbf{s})^i P' \mid R_1 \mid R).$$

□

Lemma 2 *If $P \equiv P'$ and $P \xrightarrow{\alpha} Q$, then there is Q' such that $P' \xrightarrow{\alpha} Q'$ and $Q \equiv Q'$.*

Proof The proof is by induction on the length of the derivation of $P \equiv P'$. We detail only some cases:

(SC-AUTH-DIST): Let $P = (a, \mathbf{r})(P_1 \mid Q_1)$ and $P' = (a, \mathbf{r})P_1 \mid (a, \mathbf{r})Q_1$. There are five possible transitions for P, by (LSCOPE1) − (LSCOPE5).

(LSCOPE1) : Assume that $(a, \mathbf{r})(P_1 \mid Q_1) \xrightarrow{\alpha^{(a,\mathbf{r})}} (a, \mathbf{r})P_1'$, where for $P_1 \mid Q_1 \xrightarrow{\alpha} P_1'$, and for $i = 0, 1$ it holds $\alpha \neq (a, \mathbf{s})^i a_{\mathbf{s}} l\langle \mathbf{r} \rangle$ and $\alpha \neq \tau_\omega^{(a,\mathbf{r})}$. For the transition $P_1 \mid Q_1 \xrightarrow{\alpha} P_1'$, the following three cases are possible:

- (LPAR1) : $P_1 \mid Q_1 \xrightarrow{\alpha} P_1'' \mid Q_1$ and $P_1 \xrightarrow{\alpha} P_1''$ and $\mathsf{bn}(\alpha) \notin \mathsf{fn}(Q_1)$. By (LSCOPE1) we can derive $(a, \mathbf{r})P_1 \xrightarrow{\alpha^{(a,\mathbf{r})}} (a, \mathbf{r})P_1''$. As $\mathsf{bn}(\alpha) \notin \mathsf{fn}((a, \mathbf{r})Q_1)$, by (LPAR1) we have $(a, \mathbf{r})P_1 \mid (a, \mathbf{r})Q_1 \xrightarrow{\alpha^{(a,\mathbf{r})}} (a, \mathbf{r})P_1'' \mid (a, \mathbf{r})Q_1$.
- (LCOMM1) : $P_1 \mid Q_1 \xrightarrow{\alpha} P_1'' \mid Q_1'$ and $\alpha = \tau_{\omega_1\omega_2}$ and $P_1 \xrightarrow{\alpha_{\omega_1}} P_1''$ and $Q_1 \xrightarrow{\overline{\alpha}_{\omega_2}} Q_1'$. Since α_{ω_1} and α_{ω_2} does not include unauthorized delegated roles, $(a, \mathbf{r})P_1 \mid (a, \mathbf{r})Q_1 \xrightarrow{(\tau_{\omega_1\omega_2})^{(a,\mathbf{r})}} (a, \mathbf{r})P_1'' \mid (a, \mathbf{r})Q_1'$.
- (LCOMM2) : $P_1 \mid Q_1 \xrightarrow{\alpha} (\nu c)(P_1'' \mid Q_1')$ and $\alpha = \tau_\omega$ and $P_1 \xrightarrow{\alpha_1} P_1''$ and $Q_1 \xrightarrow{\alpha_2} Q_1'$ and $\{\alpha_1, \alpha_2\} = \{(\nu c)(b, \mathbf{s})^i b_{\mathbf{s}}!l(c), (b, \mathbf{t})^j b_{\mathbf{t}}?l(c)\}$ for some $i, j \in \{0, 1\}$. By (LSCOPE1), $(a, \mathbf{r})P_1 \xrightarrow{\alpha_1^{(a,\mathbf{r})}} (a, \mathbf{r})P_1''$ and $(a, \mathbf{r})Q_1 \xrightarrow{\overline{\alpha_2}^{(a,\mathbf{r})}} (a, \mathbf{r})Q_1'$. Hence, by (LCOMM2), $(a, \mathbf{r})P_1 \mid (a, \mathbf{r})Q_1 \xrightarrow{(\tau_{\omega_1\omega_2})^{(a,\mathbf{r})}} (\nu c)((a, \mathbf{r})P_1'' \mid (a, \mathbf{r})Q_1')$.

□

Lemma 3 *If $P \to Q$ then there is Q' such that $P \xrightarrow{\tau} Q'$ and $Q \equiv Q'$.*

Proof By induction on the derivation $P \to Q$.

Case (COMM) $\delta_1(b, \mathbf{s})b_{\mathbf{s}}!l(c).P \mid \delta_2(b, \mathbf{r})b_{\mathbf{r}}?l(x).Q \to \delta_1(b, \mathbf{s})P \mid \delta_2(b, \mathbf{r})Q\{c/x\}$:

By (LOUT) and (consecutive application of) (LSCOPE1) we infer

$$\delta_1(b, \mathbf{s})b_{\mathbf{s}}!l(c).P \xrightarrow{(b,\mathbf{s})b_{\mathbf{s}}!l(c)} \delta_1(b, \mathbf{s})P,$$

and by (LIN) and (LSCOPE1) we get

$$\delta_2(b, \mathbf{r})b_{\mathbf{r}}?l(x).Q \xrightarrow{(b,\mathbf{r})b_{\mathbf{r}}?l(c)} \delta_2(b, \mathbf{r})Q\{c/x\}.$$

Therefore, by (LCOMM1), we have

$$\delta_1(b, \mathbf{s})b_{\mathbf{s}}!l(c).P \mid \delta_2(b, \mathbf{r})b_{\mathbf{r}}?l(x).Q \xrightarrow{\tau} \delta_1(b, \mathbf{s})P \mid \delta_2(b, \mathbf{r})Q\{c/x\}.$$

Case (AUTH) $\delta_1(b,\mathbf{s})(b,\mathbf{d})b_{\mathbf{s}}l\langle\mathbf{d}\rangle.P \mid \delta_2(b,\mathbf{r})b_{\mathbf{r}}l(\mathbf{d}).Q \to \delta_1(b,\mathbf{s})P \mid \delta_2(b,\mathbf{r})(b,\mathbf{d})Q$:
By (LOUT- A), (LSCOPE2) and (LSCOPE1), we get

$$\delta_1(b,\mathbf{s})(b,\mathbf{d})b_{\mathbf{s}}l\langle\mathbf{d}\rangle.P \xrightarrow{(b,\mathbf{s})(b,d)b_{\mathbf{s}}l\langle\mathbf{d}\rangle} \delta_1(b,\mathbf{s})P,$$

by (LIN- A) and (LSCOPE1),

$$\delta_2(b,\mathbf{r})b_{\mathbf{r}}l(\mathbf{d}).Q \xrightarrow{(b,\mathbf{r})b_{\mathbf{r}}l(\mathbf{d})} \delta_2(b,\mathbf{r})(b,\mathbf{d})Q,$$

and by (LCOMM1)

$$\delta_1(b,\mathbf{s})(b,\mathbf{d})b_{\mathbf{s}}l\langle\mathbf{d}\rangle.P \mid \delta_2(b,\mathbf{r})b_{\mathbf{r}}l(\mathbf{d}).Q \xrightarrow{\tau} \delta_1(b,\mathbf{s})P \mid \delta_2(b,\mathbf{r})(b,\mathbf{d})Q.$$

Case (STRU) : Assume that $P \to Q$ is derived from $P \equiv P' \to Q' \equiv Q$. By induction hypothesis, there is Q_1 such that $P' \xrightarrow{\tau} Q_1$ and $Q' \equiv Q_1$. By Lemma 2, there is Q_2 such that $P \xrightarrow{\tau} Q_2$ and $Q_2 \equiv Q_1 (\equiv Q' \equiv Q)$.

Case (PARC): If $P_1 \mid R \to Q_1 \mid R$ is derived from $P_1 \to Q_1$, by induction hypothesis there is Q_1' such that $P_1 \xrightarrow{\tau} Q_1'$ and $Q_1 \equiv Q_1'$. By (LPAR1), $P_1 \mid R \xrightarrow{\tau} Q_1' \mid R$. Since $Q_1 \equiv Q_1'$, by contextually of $\equiv$, we conclude that $Q_1 \mid R \equiv Q_1' \mid R$.

Case (NEWC) : If $(\nu a)P_1 \to (\nu a)Q_1$ is derived from $P_1 \to Q_1$, by induction hypothesis there is Q_1' such that $P_1 \xrightarrow{\tau} Q_1'$ and $Q_1 \equiv Q_1'$. By (LRES1), $(\nu a)P_1 \xrightarrow{\tau} (\nu a)Q_1'$. By contextually of $\equiv$, from $Q_1 \equiv Q_1'$, we have that $(\nu a)Q_1 \equiv (\nu a)Q_1'$. □

Lemma 4 *If $P \xrightarrow{\tau} Q$ then there is Q' such that $P \to Q'$ and $Q \equiv Q'$.*

Proof By induction on the derivation $P \xrightarrow{\tau} Q$. We have a case analysis on the last applied rule:

- (LCOMM1) : Assume that $P \mid Q \xrightarrow{\tau} P' \mid Q'$ is derived from $P \xrightarrow{\alpha} P'$ and $Q \xrightarrow{\overline{\alpha}} Q'$, where $\alpha \in \{(a,\mathbf{s})a_{\mathbf{s}}!l(b), (a,\mathbf{r})a_{\mathbf{r}}?l(b)\}$ or $\alpha \in \{(a,\mathbf{s})(a,\mathbf{d})a_{\mathbf{s}}l\langle\mathbf{d}\rangle, (a,\mathbf{r})a_{\mathbf{r}}l(\mathbf{d})\}$. By Lemma 1, up to $\equiv$,

  $$P, Q \in \{(\nu\vec{a}_1)(\delta_1(a,\mathbf{s})a_{\mathbf{s}}!l(b).P_1' \mid R), (\nu\vec{a}_2)(\delta_2(a,\mathbf{r})a_{\mathbf{r}}?l(x).P_2' \mid R')\} \text{ or}$$

  and we get the proof by (COMM), or

  $$P, Q \in \{(\nu\vec{a}_1)(\delta_1(a,\mathbf{s})(a,\mathbf{d})a_{\mathbf{s}}l\langle d\rangle.P' \mid R), (\nu\vec{a}_2)(\delta_2(a,\mathbf{r})a_{\mathbf{r}}l(d).P' \mid R)\}$$

  and the proof is completed by (AUTH).
- (LCOMM2) : Assume that $P \mid Q \xrightarrow{\tau} P' \mid Q'$ is derived from $P \xrightarrow{(\nu a)(b,\mathbf{s})b_{\mathbf{s}}!l(a)} P'$ and $Q \xrightarrow{(b,\mathbf{r})b_{\mathbf{r}}?l(a)} Q'$. By Lemma 1,

  $$P \equiv (\nu\vec{a}_1)(\nu a)(\delta_1(b,\mathbf{s})b_{\mathbf{s}}!l(a).P_1' \mid R) \text{ and } Q \equiv (\nu\vec{a}_2)(\delta_2(b,\mathbf{r})b_{\mathbf{r}}?l(x).P_2' \mid R').$$

  we get the proof by (SC-REST-EXTR) and (COMM).
- (LSCOPE5) : Assume that $(a,\mathbf{d})P \xrightarrow{\tau} Q$ is derived from $P \xrightarrow{\tau^{(a,\mathbf{d})}} Q$. By Lemma 1,

  $$P \equiv (\nu\vec{a}_1)(\delta_1(a,\mathbf{s})a_{\mathbf{s}}l\langle d\rangle.P' \mid \delta_2(a,\mathbf{r})a_{\mathbf{r}}l(d).P' \mid R) \text{ and } (a,\mathbf{d}) \notin \delta_1,$$

  concluding the proof by (SC-AUTH-SCOPE) and (AUTH).
- (LRES1) : Assume that $(\nu a)P \xrightarrow{\tau} (\nu a)Q$ is derived from $P \xrightarrow{\tau} Q$. By induction hypothesis there is Q' such that $P \to Q'$ and $Q' \equiv Q$. We get the proof by (NEWC) and contextuality of $\equiv$.
- (LPAR1) : Assume that $P \mid R \xrightarrow{\tau} Q \mid R$ is derived from $P \xrightarrow{\tau} Q$. By induction hypothesis there is Q' such that $P \to Q'$ and $Q' \equiv Q$. We get the proof by (PARC) and contextuality of $\equiv$.

□

Proposition 2 *$P \to Q$ if and only if there is Q' such that $P \xrightarrow{\tau} Q'$ and $Q \equiv Q'$.*

Proof The proof follows by Lemma 3 and 4. □

Proposition 3 *If $P \equiv Q$ then $P \sim Q$.*

Proof Let us prove that $\equiv \subseteq \sim$.

Assume $P \equiv Q$ and $P \xrightarrow{\alpha} P'$. Then by Lemma 3 there is a process Q' such that $Q \xrightarrow{\alpha} Q'$ and $Q \equiv Q'$. Which proves $\equiv$ to be a strong bisimulation. Therefore $\equiv \subseteq \sim$, since $\sim$ is the largest strong bisimulation. □

Proposition 4 *If $P \sim Q$ then $\mathcal{C}[P] \sim \mathcal{C}[Q]$.*

Proof Assume that $\sim^c$ is a contextual congruence closure (relation) of $\sim$, i.e.

$P \sim Q$ then $\mathcal{C}[P] \sim^c \mathcal{C}[Q]$.

Then

$\sim \subseteq \sim^c$

which follows by taking $\mathcal{C}[\cdot] = \cdot$. Also, since $\sim^c$ is a contextual closure of $\sim$ then $P \sim^c Q$ implies that $P = \mathcal{C}[P_1]$, $Q = \mathcal{C}[Q_1]$ and $P_1 \sim Q_1$, for some context $\mathcal{C}[\cdot]$ and some processes P_1 and Q_1.

Let us show that $\sim^c$ is a strong bisimulation, that is
if $P \sim^c Q$ and $P \xrightarrow{\alpha} P'$, then there is Q' such that $Q \xrightarrow{\alpha} Q'$ and $P' \sim^c Q'$.
Assume $P \sim^c Q$ and $P \xrightarrow{\alpha} P'$, where $P = \mathcal{C}[P_1]$, $Q = \mathcal{C}[Q_1]$ and $P_1 \sim Q_1$.
The proof is by induction on the structure of the context $\mathcal{C}[\cdot]$.

- If $\mathcal{C}[\cdot] = \cdot$, then $P = P_1$, $Q = Q_1$, hence $P \sim Q$ and the statement holds.
- If $\mathcal{C}[\cdot] = (a, \mathbf{r})\mathcal{C}_1[\cdot]$, then $P = (a, \mathbf{r})\mathcal{C}_1[P_1]$, $Q = (a, \mathbf{r})\mathcal{C}_1[Q_1]$ and $P_1 \sim Q_1$. For the transition $(a, \mathbf{r})\mathcal{C}_1[P_1] \xrightarrow{\alpha} P'$ there are five possible cases: (LSCOPE1)-(LSCOPE5). We show only case (LSCOPE1) : then $(a, \mathbf{r})\mathcal{C}_1[P_1] \xrightarrow{(a,\mathbf{r})\cdot\alpha} (a, \mathbf{r})P'$, where $\mathcal{C}_1[P_1] \xrightarrow{\alpha} P'$. Then by the induction hypothesis $\mathcal{C}_1[Q_1] \xrightarrow{\alpha} Q'$ and $P' \sim^c Q'$, for some Q'. The proof is completed by $(a, \mathbf{r})\mathcal{C}_1[Q_1] \xrightarrow{(a,\mathbf{r})\cdot\alpha} (a, \mathbf{r})Q'$ and $(a, \mathbf{r})P' \sim^c (a, \mathbf{r})Q'$ since $\sim^c$ is a contextual closure. The case $\mathcal{C}[\cdot] = R \mid \mathcal{C}_1[\cdot]$ can be proved similarly.
- If $\mathcal{C}[\cdot] = (\nu a)\mathcal{C}_1[\cdot]$, then $P = (\nu a)\mathcal{C}_1[P_1]$, $Q = (\nu a)\mathcal{C}_1[Q_1]$ and $P_1 \sim Q_1$.
For the transition $(\nu a)\mathcal{C}_1[P_1] \xrightarrow{\alpha} P'$ there are two cases: (LRES1) and (LRES2). For (LRES1) the statement follows by the induction hypothesis. For (LRES2) by Lemma 1(i) it follows that
$P \equiv (\nu \vec{a}_1)(\nu c)(\delta(b, \mathbf{s})^i b_{\mathbf{s}}!l(c).R' \mid R)$ and $P' \equiv (\nu \vec{a}_1)(\delta(b, \mathbf{s})^i R' \mid R)$.

The relation $\sim^c$ is a strong bisimulation, then $\sim^c \subseteq \sim$ since $\sim$ is the largest bisimulation. To this end $\sim^c = \sim$, hence the strong bisimilarity $\sim$ is a congruence with respect to active contexts. □

A.2. Proofs from Sect. 3

Lemma 5 (Inversion Lemma)

1. *If* $\Delta;\ \Gamma \vdash_\emptyset 0$ *then* $\Delta = \Delta_{\mathbf{end}}$.
2. *If* $\Delta;\ \Gamma \vdash_\Sigma (\nu a)P$, *then*
   (a) $\Delta;\ \Gamma, a : l(B) \vdash_\Sigma P$, *or*
   (b) $\Delta;\ a : B;\ \Gamma \vdash_\Sigma P$ *and* $\mathtt{matched}(B)$ *and* $a \notin \mathtt{pr}_1(\Sigma)$.
3. *If* $\Delta;\ \Gamma \vdash_\Xi (a, \mathbf{r})P$ *then* $\Delta, \Gamma \vdash_\Sigma P$ *and* $\Xi = \Sigma \setminus \{(a, \mathbf{r})\}$.
4. *If* $\Delta;\ \Gamma \vdash_\Xi a_{\mathbf{r}}\, l(\mathbf{s}).P$ *then* $\Delta = \Delta' \circ a : B'$ *and* $\Delta' \circ a : B;\ \Gamma \vdash_\Sigma P$ *and* $?\mathbf{r}\, l(\mathbf{s}).B <: B'$ *and* $\Xi = (\Sigma \setminus \{(a, \mathbf{s})\}) \cup \{(a, \mathbf{r})\}$
5. *If* $\Delta;\ \Gamma \vdash_\Xi a_{\mathbf{r}}\, l\langle \mathbf{s} \rangle.P$ *then* $\Delta = \Delta' \circ a : B'$ *and* $\Delta' \circ a : B;\ \Gamma \vdash_\Sigma P$ *and* $!\mathbf{r}\, l(\mathbf{s}).B <: B'$ *and* $\Xi = \Sigma \cup \{(a, \mathbf{r}), (a, \mathbf{s})\}$ *and* $(a, \mathbf{s}) \notin \Sigma$.
6. *If* $\Delta;\ \Gamma \vdash_\Xi a_{\mathbf{r}}?l(b).P$ *then*
   (a) $\Delta = \Delta' \circ a : B''$ *and* $\Delta' \circ a : B, b : B';\ \Gamma \vdash_\Sigma P$ *and* $?\mathbf{r}\, l(B').B <: B''$ *and* $b \notin \mathtt{pr}_1(\Sigma)$ *and* $\Xi = \Sigma \cup \{(a, \mathbf{r})\}$, *or*

(b) $\Delta = \Delta' \circ a : B'$ *and* $\Delta' \circ a : B$; $\Gamma', b : T \vdash_\Sigma P$ *and* $?\mathbf{r}\, l(T).B <: B'$ *and* $b \notin \mathtt{pr}_1(\Sigma)$ *and* $\Xi = \Sigma \cup \{(a, \mathbf{r})\}$, *or*

(c) $\Gamma = \Gamma', a : l(B)$ *and* $\Delta \circ b : B$; $\Gamma \vdash_\Sigma P$ *and* $b \notin \mathtt{pr}_1(\Sigma)$ *and* $\Xi = \Sigma \cup \{(a, \mathbf{r})\}$.

7. *If* Δ; $\Gamma \vdash_\Xi a_\mathbf{r}!l(b).P$, *then*

(a) $\Delta = \Delta' \circ a : B'' \circ b : B'$ *and* $\Delta' \circ a : B$; $\Gamma \vdash_\Sigma P$ *and* $!\mathbf{r}\, l(B').B <: B''$ *and* $\Xi = \Sigma \cup \{(a, \mathbf{r})\}$, *or*

(b) $\Delta = \Delta' \circ a : B'$ *and* $\Gamma = \Gamma', b : T$ *and* $\Delta' \circ a : B$; $\Gamma \vdash_\Sigma P$ *and* $!\mathbf{r}\, l(T).B <: B'$ *and* $\Xi = \Sigma \cup \{(a, \mathbf{r})\}$.

(c) $\Delta = \Delta' \circ b : B$ *and* $\Gamma = \Gamma', a : l(B)$ *and* Δ'; $\Gamma \vdash_\Sigma P$ *and* $\Xi = \Sigma \cup \{(a, \mathbf{r})\}$.

8. *If* Δ; $\Gamma \vdash_{\Sigma'} P \mid Q$, *then* $\Delta = \Delta_1 \circ \Delta_2$ *and* $\Sigma' = \Sigma \cup \Xi$ *and* Δ_1; $\Gamma \vdash_\Sigma P$ *and* Δ_2; $\Gamma \vdash_\Xi Q$ *for some* Σ, Ξ, Δ_1 *and* Δ_2.

Lemma 6 (Substitution lemma) *Let* Δ; $\Gamma \vdash_\Sigma P$ *for some* Δ *and* Γ.

(a) *If* $\Delta = \Delta', c : B$ *and* $\Delta' \circ b : B$ *is defined then* $\Delta' \circ b : B$; $\Gamma \vdash_\Sigma P\{b/c\}$.

(b) *If* $\Gamma = \Gamma', c : T$ *and* $\Gamma', b : T$ *is defined then* Δ; $\Gamma', b : T \vdash_\Sigma P\{b/c\}$.

Lemma 7 (Subject congruence) *If* Δ; $\Gamma \vdash_\Sigma P$ *and* $P \equiv Q$ *then* Δ; $\Gamma \vdash_\Sigma Q$.

Theorem 1 (Type preservation) *Let* Δ; $\Gamma \vdash_\Sigma P$ *for some* Δ, Γ, Σ *and* P. *If* $P \to Q$ *then there is* Δ' *such that* $\Delta \to \Delta'$ *and* Δ'; $\Gamma \vdash_\Sigma Q$.

Proof The proof is by induction on the of derivation $P \to Q$ and by cases on the last rule applied. The only interesting case is induced by (AUTH).
In that case $P = \delta_1(b, \mathbf{s})(b, \mathbf{d})b_\mathbf{s} l\langle\mathbf{d}\rangle.P' \mid \delta_2(b, \mathbf{r})b_\mathbf{r} l(\mathbf{d}).Q'$ and $Q = \delta_1(b, \mathbf{s})P' \mid \delta_2(b, \mathbf{r})(b, \mathbf{d})Q'$.
From Lemma 5(8) we obtain that for some Δ_1, Δ_2, Σ_1 and Σ_2:

- $\Delta = \Delta_1 \circ \Delta_2$
- $\Sigma = \Sigma_1 \cup \Sigma_2$

(1) $\Delta_1, \Gamma \vdash_{\Sigma_1} \delta_1(b, \mathbf{s})(b, \mathbf{d})b_\mathbf{s} l\langle\mathbf{d}\rangle.P'$

(2) $\Delta_2, \Gamma \vdash_{\Sigma_2} \delta_2(b, \mathbf{r})b_\mathbf{r} l(\mathbf{d}).Q'$

From (1) and (2) and Lemma 5(3) we obtain:

(3) $\Delta_1, \Gamma \vdash_{\Sigma_1'} b_\mathbf{s} l\langle\mathbf{d}\rangle.P'$ where $\Sigma_1 = \Sigma_1' \setminus \{\delta_1, (b, \mathbf{s}), (b, \mathbf{d})\}$

(4) $\Delta_2, \Gamma \vdash_{\Sigma_2'} b_\mathbf{r} l(\mathbf{d}).Q'$ where $\Sigma_2 = \Sigma_2' \setminus \{\delta_2, (b, \mathbf{r})\}$

From (3) and Lemma 5(5) we obtain:

- $\Delta_1 = \Delta_1' \circ b : B_1$

(5) $\Delta_1' \circ b : B_1'$; $\Gamma \vdash_{\Sigma_1''} P'$ where $\Sigma_1' = \Sigma_1'' \cup \{(b, \mathbf{s}), (b, \mathbf{d})\}$

- $!\mathbf{s}\, l(\mathbf{d}).B_1' <: B_1$
- $(b, \mathbf{d}) \notin \Sigma_1''$

From (4) and Lemma 5(4) we obtain:

- $\Delta_2 = \Delta_2' \circ b : B_2$

(6) $\Delta_2' \circ b : B_2'$; $\Gamma \vdash_{\Sigma_2''} Q'$ where $\Sigma_2' = (\Sigma_2'' \setminus \{(b, \mathbf{d})\}) \cup \{(b, \mathbf{r})\}$

- $?\mathbf{r}\, l(\mathbf{d}).B_2' <: B_2$

From (5),(6) and (T-AUTH) we obtain:

- $\Delta_1' \circ b : B_1'$; $\Gamma \vdash_{\Sigma_1'''} \delta_1(b, \mathbf{s})P'$ where $\Sigma_1''' = \Sigma_1'' \setminus \{\delta_1, (b, \mathbf{s})\}$
- $\Delta_2' \circ b : B_2'$; $\Gamma \vdash_{\Sigma_2'''} \delta_2(b, \mathbf{r})(b, d)Q'$ where $\Sigma_2''' = \Sigma_2'' \setminus \{\delta_2, (b, \mathbf{r}), (b, d)\}$

Finally, since one can easily derive $\Sigma_1''' \cup \Sigma_2''' = \Sigma$ and $B_1 \circ B_2 \to B_1' \circ B_2'$, from previous and (T-PAR) we can conclude Δ'; $\Gamma \vdash_\Sigma Q$ and $\Delta \to \Delta'$ where $\Delta' = \Delta_1' \circ b : B_1' \circ \Delta_2' \circ b : B_2' = \Delta_1' \circ \Delta_2' \circ b : B_1' \circ B_2'$. □

Lemma 8 *If* Δ; $\Gamma \vdash_\emptyset \mathcal{C}[\alpha_{(a,\mathbf{r})}.Q]$ *then* $auth(\mathcal{C}[\cdot], (a, \mathbf{r})) = true$ *and if* $\alpha_{(a,\mathbf{r})} = a_\mathbf{r} l\langle\mathbf{d}\rangle$ *then* $auth(\mathcal{C}[\cdot], (a, \mathbf{d})) = true$.

Proposition 5 *If* P *is a well-typed process, then* P *is not an authorization error.*

References

[BCCDC11] Bono V, Capecchi S, Castellani I, Dezani-Ciancaglini M (2011) A reputation system for multirole sessions. In: Roberto B, Vladimiro S (eds) Trustworthy Global Computing—6th International Symposium, TGC 2011, Aachen, Germany, June 9-10, 2011. Revised Selected Papers, vol. 7173 of Lecture Notes in Computer Science. Springer, pp 1–24

[BCD+15] Bartoletti M, Castellani I, Deniélou P, Dezani-Ciancaglini M, Ghilezan S, Pantovic J, Pérez JA, Thiemann P, Toninho B, Vieira HT (2015) Combining behavioural types with security analysis. J Log Algebr Meth Program, 84(6):763–780

[BCG05] Bonelli E, Compagnoni AB, Gunter EL (2005) Correspondence assertions for process synchronization in concurrent communications. J Funct Program 15(2):219–247

[BCVV12] Baltazar P, Caires L, Vasconcelos VT, Vieira HT (2012) A type system for flexible role assignment in multiparty communicating systems. In: Catuscia P and Mark Dermot R (eds) Trustworthy Global Computing—7th International Symposium, TGC 2012, Revised Selected Papers, Vol 8191 of Lecture Notes in Computer Science. Springer, pp 82–96

[CCDC11] Capecchi S, Castellani I, Dezani-Ciancaglini M (2011) Information flow safety in multiparty sessions. In: Bas L and Frank V (eds) Proceedings 18th International Workshop on Expressiveness in Concurrency, EXPRESS 2011, Aachen, Germany, 5th September 2011, Vol 64 EPTCS, pp 16–30

[CCDCR10] Capecchi S, Castellani I, Dezani-Ciancaglini M, Rezk T (2010) Session types for access and information flow control. In: Paul G, François L (eds) CONCUR 2010—Concurrency Theory, 21th International Conference, CONCUR 2010, Paris, France, August 31–September 3, 2010. Proceedings, Vol 6269 of Lecture Notes in Computer Science, Springer, pp 237–252

[CPN98] David G, Clarke, Potter J, Noble J (1998) Ownership types for flexible alias protection. In: Bjørn N. Freeman-Benson and Craig Chambers (eds) Proceedings of the 1998 ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA '98), Vancouver, British Columbia, Canada, October 18-22, 1998. ACM, pp 48–64

[CV10] Caires L, Vieira HT (2010) Conversation types. Theor Comp Sci 411(51–52):4399–4440

[DGJP10] Dezani-Ciancaglini M, Ghilezan S, Jaksic S, Pantovic J (2010) Types for role-based access control of dynamic web data. In Julio Mariño (ed) Functional and Constraint Logic Programming—19th International Workshop, WFLP 2010, Madrid, Spain, January 17, 2010. Revised Selected Papers, volume 6559 of *Lecture Notes in Computer Science*. Springer, pp 1–29

[DY11] Pierre-Malo D, Yoshida N (2011) Dynamic multirole session types. In: Thomas B, Mooly S (eds) Proceedings of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2011, Austin, TX, USA, January 26-28, 2011, ACM, pp 435–446

[FGM07] Fournet C, Gordon AD, Maffeis S (2007) A type discipline for authorization policies. ACM Trans Program Lang Syst, 29(5)

[GJP+14] Ghilezan S, Jaksic S, Pantovic J, Pérez JA, Vieira HT (2014) Dynamic role authorization in multiparty conversations. In: Proceedings of BEAT 2014, Vol. 162 of EPTCS, pp 1–8

[GJP+15] Ghilezan S, Jaksic S, Pantovic J, Pérez JA, Vieira HT (2016) A typed model for dynamic authorizations. In: Gay S, Alglave J (eds) Proceedings Eighth International Workshop on Programming Language Approaches to Concurrency- and Communication-cEntric Software, London, 18th April 2015. Electronic Proceedings in Theoretical Computer Science, vol 203. Open Publishing Association, pp 73–84. doi:10.4204/EPTCS.203.6

[GP09] Gorla D, Pugliese R (2009) Dynamic management of capabilities in a network aware coordination language. J Log Algebr Program 78(8):665–689

[GPV12] Giunti M, Palamidessi C, Valencia FD (2012) Hide and new in the pi-calculus. In: Proceedings Combined 19th International Workshop on Expressiveness in Concurrency and 9th Workshop on Structured Operational Semantics, EXPRESS/SOS 2012, volume 89 of EPTCS, pp 65–79

[HLV+16] Huttel H, Lanese I, Vasconcelos VT, Caires L, Carbone M, Pierre-Malo D, Mostrous D, Padovani L, Ravara A, Tuosto E, Vieira HT, Zavattaro G (2016) Foundations of behavioural types. ACM Comput. Surv. To appear. Preliminary version available at http://www.behavioural-types.eu/publications/.

[HRU76] Michael A, Harrison, Walter L, Ruzzo, Jeffrey D (1976) Ullman. Protection in operating systems. Commun ACM 19(8):461–471

[Lam74] Lampson BW (1974) Protection. Operating Syst Rev 8(1):18–24

[LPT07] Lapadula A, Pugliese R, Tiezzi F (2007) Regulating data exchange in service oriented applications. In Farhad Arbab and Marjan Sirjani, editors, International Symposium on Fundamentals of Software Engineering, International Symposium, FSEN 2007, Tehran, Iran, April 17-19, 2007, Proceedings, volume 4767 of Lecture Notes in Computer Science. Springer, pp 223–239

[San92] Sandhu RS (1992) The typed access matrix model. In 1992 IEEE Computer Society Symposium on Research in Security and Privacy, Oakland, CA, USA, May 4–6, 1992. IEEE Computer Society, pp 122–136

[SCC10] Swamy N, Chen J, Chugh R (2010) Enforcing stateful authorization and information flow policies in fine. In: Programming Languages and Systems, 19th European Symposium on Programming, ESOP 2010, Proceedings, Vol 6012 of Lecture Notes in Computer Science, Springer, pp 529–549

[SdV00] Samarati P, De Capitani di Vimercati S (2000) Access control: Policies, models, and mechanisms. In: Riccardo Focardi, Roberto Gorrieri (eds) Foundations of Security Analysis and Design, Tutorial Lectures [revised versions of lectures given during the IFIP WG 1.7 International School on Foundations of Security Analysis and Design, FOSAD 2000, Bertinoro, Italy, September 2000], Vol. 2171 of Lecture Notes in Computer Science. Springer, pp 137–196

[SW01] Sangiorgi D, Walker D (2001) The Pi-Calculus—a theory of mobile processes. Cambridge University Press

[VY02] Vivas J, Yoshida N (2002) Dynamic channel screening in the higher order pi-calculus. Electr Notes Theor Comput Sci 66(3):170–184

Received 9 March 2015

Revised 14 November 2015

Accepted 22 January 2016 by Thomas Hildebrandt, Joachim Parrow, Matthias Weidlich, and Marco Carbone

Published online 18 March 2016